AN IMPLEMENTATION OF A SPATIAL DATABASE SYSTEM

By

SUBRAMANIAN SIVARAMAKRISHNAN

Bachelor of Engineering

Madurai Kamaraj University

Madurai, India

1990

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December 1993

AN IMPLEMENTATION OF A SPATIAL

DATABASE SYSTEM

Thesis Approved:

Huizhu Lu

Thesis Adviser

M. Samadzadeh-H.

Blayne E. Mayfield

Thomas C. Collins

Dean of the Graduate College

ACKNOWLEDGMENTS

I wish to express my sincere appreciation to my major advisor, Dr. Huizhu Lu for her intelligent supervision, constructive guidance, and advice. I would also like to thank Dr. Mansur Samadzadeh and Dr. Blayne Mayfield for serving on my committee and providing guidance, assistance, and encouragement.

Moreover, I would like to thank the Computer Science Department, Oklahoma State University, for giving me a teaching assistantship position during the course of my graduate studies.

I would like to thank my parents who motivated and supported me all the way during my course of study. To my brother Rajan, my sister Vani, and Dr. Ravi Ramnath, I thank them for giving me advice, moral support, and encouragement.

TABLE OF CONTENTS

Chapter Page

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Spatial data is a representation of a location in a particular space. Point data is a location with zero size. A location with non-zero size occupies space. Spatial database contains data of n-dimensions with information about objects, their extent, and position in space. Hence it requires data structures to represent the spatial data. These spatial index structures will provide efficient capabilities for retrieval and updating. It is used in various applications such as computer-aided design (CAD), very large scale integrated (VLSI) design, image processing, robotics, and geographic information systems (GIS) [BEN91, GUE90].

"Current database systems support conventional indices such as B-tree, Indexed Sequential Access Method (ISAM), and hashing techniques, but they do not provide efficient spatial data indexing" [ULL88]. So, accessing a particular spatial object is constrained to be a linear search on the set of objects.

A spatial indexing structure can be built using high-level objects in a spatial database system. Different indexing techniques can be used in this database system depending on the application platform that is being used. Various index structures have been compared [BEC90, BLA90, DAN85, DAN86, FAL87, GAR82, GUE90,

GUT84, HUT90, OHS83, OHS90, SAM89, SAM90, SEL87] and the most appropriate structure chosen is the R*-tree. The R*-tree belongs to the family of R-trees and these come under the organization of bounding rectangles for spatial objects. The R*-tree uses the principles of a B-tree and can handle point and spatial data easily. Other index structures like inverted files are useful for static databases. The balanced and nested grid file is an improvement over the grid file but requires a higher storage utilization. In quadtrees and k-d-trees, restructuring is expensive if the distribution of data is non-uniform. K-d-B trees are a combination of the properties of B-trees and K-d trees. K-d-B trees are not guaranteed to occupy half of the pages in the tree.

Spatial objects involve large amounts of data to be managed for efficient organisation and retrieval. Hence, a simple spatial database system can be designed to support operations associated with spatial data. This approach could both offer a more flexible model for dealing with complex and huge data and also reduce the semantic gap between real-world objects and their corresponding abstractions.

Research Objective

The objective of the thesis is to design and implement a prototype spatial database system that can handle basic operations such as searching, retrieval, insertion, and deletion within a given spatial region. A set of VLSI data is used as an application of the system. The validity of the system is observed through the

application.

Scope of Work

The R*-tree is chosen as the index structure for the implementation after comparison with different index structures. It is an enhanced modification of the R-tree; these types of tree structures come under the category of an approximate optimization of the area of the bounding rectangle in each node. The average cost of insertion in an R*-tree is less than the cost of insertion in an R-tree [BEC90].

The R*-tree has a complex index structure with additional conditions to be met to make it a better and more efficient R-tree. The index structure can handle various operations such as insertion, deletion, access, and searching for a geometric application. An R-tree index structure which is in general less complex than an R*-tree was built and modifications were done to suit the requirements of an R*-tree. In order to achieve the stated objective, the R*-tree index structure was first validated with a standard test bed to confirm that that different operations can be safely performed. The test was done by randomly generating rectangles or coordinates for spatial objects within a particular bounding region. Subsequently, a geometric application such as a VLSI design layout was used to confirm the operations that were implemented. The VLSI design was converted into an ordinary text file consisting of rectangular coordinates representing spatial regions. This spatial database system was developed using the C programming language because of its

versatility, ease of use, and harmony in blending itself well with the spatial index system. The rectangular coordinates were retrieved from the application input file and the R*-tree spatial index structure was formed. Similar spatial regions were accessed when searching for certain areas in the geometric application. Hence, a database management system was developed to handle simple operations such as insertion, deletion, searching, and accessing on the geometric application (two dimensional space). The performance of the system was validated through the application.

The data that were used for the spatial index structure was an application consisting of a VLSI design. The coordinates were obtained by conversion of a VLSI design layout file into a text file with information about the lower and upper x and y axes. The leaf level of the R*-tree pointed to these rectangular coordinate locations or spatial objects. Hence, efficient searching could be performed based on the coordinates in the region.

If such a system is developed as a commercial package, it should have data management capabilities with increased functionality and support for spatial objects. Otherwise, there would be a lot of time and effort required to obtain information from such very large databases [GUE90].

The organization of the thesis is as follows. Chapter II provides a literature survey on some of the spatial index structures. A complete description of the algorithms used for implementation of the R-tree is listed along with details on the execution of the program. Chapter III gives an introduction to the steps taken to

implement the spatial database system. A detailed description of the algorithms used for the implementation of the R*-tree is given along with an explanation of the program execution using random number generation initially, and later, execution of program using VLSI data application to test for performance. Chapter IV explains the results obtained from the spatial database system that validate the spatial index system with the geometric application. Chapter V concludes with a summary of the thesis, and a note on future research in this field.

CHAPTER II

LITERATURE REVIEW

A database is an organized integrated collection of data. The purpose of a database is to maintain data with respect to availability, integrity, and security. A relational database is a database that is perceived by the user as a collection of time-varying, normalized relations of various degrees [DAT92]. It is based on the concepts of data abstraction, sharing, and modularity. A spatial database system can be built by first constructing a database management system. This is accomplished by building an index structure for the spatial database on index records containing geometric objects. Various index structures are reviewed in this chapter with emphasis on the R-tree.

Spatial Data Structures

To handle spatial objects, a database system requires an indexing mechanism to retrieve data items efficiently based on their spatial locations. There are a number of spatial index structures that can handle their chosen application respectively [SAM90]. The goal of a spatial data structure is to formulate a method for dividing

the entire space into cells, and to provide a mapping between these cells and the region in space occupied by an object [NIE89].

A simple query into a spatial database is to locate all objects that enclose a given point. This is known as a point search. In range search, objects are located that overlap a given search space. Bentley and Friedman in their article [BEN79] explain the definition of range searching. In database terminology, a collection of records is a file. Each record contains several attributes or keys. A query requests to produce all records that satisfy certain predefined characteristics. An orthogonal range query requests records with key values, each between specified upper and lower limits. Range searching is the process of retrieving such kind of records. Geometrically speaking, the record attributes can be regarded as coordinates where the k values for each record represents a point in k-dimensional coordinate space. The file of records consists of a set of points in k-space. A range query forms a k-dimensional hyperrectangle in space or a box and all points lying inside this hyperrectangle are to be searched.

A spatial index structure is used to do such operations so that the user can efficiently access the objects in a certain spatial neighborhood. There are three basic methods which broadly classify the way spatial index structures manage spatial objects [SEE90].

(1) Clipping where the objects are divided along the partitioning lines of the underlying access structure.

(2) Overlapping regions where each partition of the access structure may contain

any object it overlaps.

(3) Transformation where extended spatial objects are mapped into higher dimensional points.

An example for each of the above methods is given. Transformation is used for simple objects such as polygons. Arbitrary polygons are usually represented by their bounding boxes. Overlapping region method includes structures like an R-tree. An example of the clipping scheme is a cell tree [GUN89].

There are many spatial data structures that are hierarchical. That is, they are based on the principle of recursive decomposition (similar to divide and conquer methods) [AHO74]. Some examples are quadtrees, octrees, k-d trees, binary image trees [GAR82, SAM89]. The other types of spatial data structures are R-trees, R^+-trees, R^*-trees [BEC90, FAL87, GUT84, SEL87], which deal with collections of small rectangles, and grid file and BANG file [FRE87], which come under the technique of bucket methods, as opposed to tree structures where pointers are inevitable when the data are stored in external storage.

There are different types of file structures for retrieval of multiple keys in a file. In an inverted file, the inverted lists which are stored on disk must be searched and retrieved in order to find all pertinent records, then these records must be retrieved. The number of block transfers to and from secondary storage is used as a measure of efficiency in retrieval and update operations. Hence, the inverted file is used for applications having large static databases. Dynamic multidimensional file structures (as opposed to static ones) are those whose records can be inserted or

deleted interactively through queries. Searches can also be intermixed and no periodic reorganization is required. Secondary index files and multikey hashing file structures require that the index and records be stored in secondary storage. This leads to a large number of disk accesses [NIE84].

Grid File

In multidimensional search structures, where the embedding space method is used, an overfull bucket results in a predetermined split of the space in an attempt to remedy an overfull situation. The grid file uses this kind of method. In the main memory, a linear scale for each attribute is stored, one for each dimension, to define the positions of the grid regions or grid partitions of the data space. The secondary storage contains the grid directory and data buckets. Hence, at most two disk accesses are required for exact match queries. The bucket utilization is reasonable.

As the grid directory becomes large, a two-level directory may give rise to improved performance. This leads to increased disk accesses, split and merge operations. But in certain cases where several adjacent regions are pointed at by the same bucket, overflow occurs and causes non-uniform distribution of data.

Balanced and Nested Grid File

The balanced and nested grid (BANG) file is an improvement over the grid

file. It utilizes the embedded space approach. It partitions the space into block regions by successive binary divisions [FRE87]. The growth of the grid directory is constrained by including additional information with each grid directory entry. The directory entry in a BANG file is represented as follows:

(r, l)

where r is the region number and l is the level number of the entry. These numbers determine a distinct subregion which is used when splitting occurs. In addition, the BANG file allows nested block regions and determines the optimum splitting dimension during each split. Thus redistribution of regions for balancing the distribution of data points is possible for non-uniform data distributions.

Compared to the grid file, the BANG file has fewer disk accesses, splitting and merging [FRE87]. Yet it requires a higher storage utilization [FRE87]. It can handle a non-point data by representing a point in a higher dimension.

Quadtree

The quadtree consists of nodes in which intermediate nodes have four children and the rest point to leaves [FIN74, GAR82]. Each node is stored as a record with four quadrants, namely north-west, north-east, south-west and south-east, and a field to identify the color used. Quadtrees are hierarchical structures for storing point data. Restructuring is expensive if the distribution of data is non-uniform. Quadtrees have many null pointers even in a balanced tree structure [SAM84, SAM89].

K-d-tree

A single-key binary search performs three functions [BEN79]. It stores the records of a file, divides the data space into segments by using discriminators and it gives a directory among the segments (a tree structure). A K-d tree is a generalization of a binary search tree to multiple dimensions with the range [1..k] [BEN75]. Each record in a file that needs to be represented as a K-d tree is stored as a node in the tree. Each node contains two pointers which are either null or point to another node in a subtree. Each node has a discriminator which is an integer between 0 and k-1, inclusive. At any given level of the tree, the nodes in that level have the same discriminator value. Thus the value of the discriminator at the root level is 0, its two sons have a value of 1. The k^{th} level has the discriminator value k-1. The value becomes 0 again when the $(k+1)^{th}$ level is reached. Therefore, the values of the discriminator are cyclic based on the number of dimensions in the tree. Figure 1 gives an illustration of the above definitions [BEN75]. Point data (A through G) are represented as coordinates in two dimensional space. Figure 1 gives a tree representation of the above coordinates. The empty boxes represent null children.

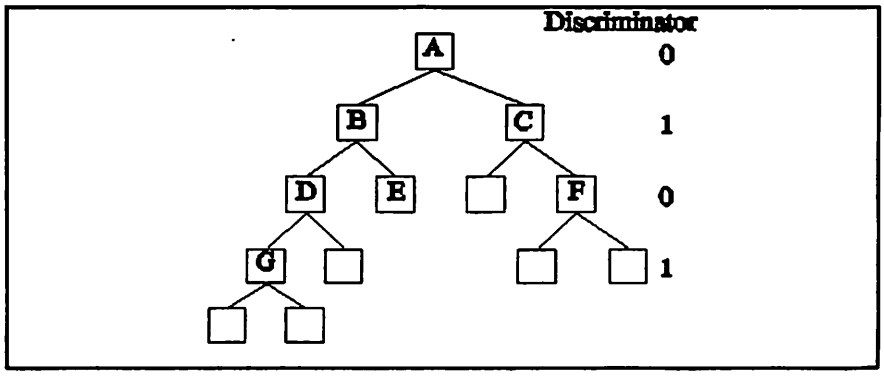

Figure 1. An example of a K-d-tree.

The discriminator need not be stored as a field in each node because it can be easily calculated based on the level and the value of k. It is observed that k-d trees are hierarchical structures for storing point data. Restructuring is costly for non-uniform distribution of data just as in a quadtree. K-d-trees require many disk accesses during an access or update [SAM89].

K-d-B-tree

B-trees and K-d-trees are combined to form K-d-B-trees.The leaf nodes of the K-d-B-tree consist of pointer pages that point to those records which correspond to a region in k-dimensional space. The intermediate nodes are region pages that show the partitioning of a region into non-overlapping, jointly exhaustive subregions. The root of the tree reflects the first partitioning of the k-dimensional space. Efficient utilization of I/O channels is obtained by requiring pointer and region pages to be approximately the size of blocks in secondary memory [ROB81].

R-tree

An R-tree is similar to a B-tree. It is a multi-level, height-balanced tree structure designed to handle n-dimensional objects. The records are situated in the leaf nodes which contain pointers to spatial data objects [GUT84]. Intermediate nodes on a given level can overlap; hence their rectangles do not represent disjoint

regions. The R-tree is a dynamic index structure since operations such as insertion and deletion can be intermingled with queries.

The R-tree has the following structure. Leaf nodes consist of the following pair of entries

(I, tuple-id)

where tuple-id is a pointer to a record or spatial object, and I is an n-dimensional rectangle which encloses the spatial object. Non-leaf nodes contain the following pair of entries

(I, child-pointers)

where child pointers are pointers to children of a particular node, and I is the bounding box or smallest rectangle that covers all the rectangles in the entries of its children. If the maximum number of entries is M, then

$m \leq M/2$

where m is the minimum number of entries in a node.

Properties

An R-tree has the following properties [GUT84]:

(1) The root has at least two children unless it is a leaf.

(2) All leaves appear on the same level.

(3) Each non-leaf node has between m and M children unless it is the root.

(4) Each leaf node has between m and M index records unless it is the root.

An example to illustrate an R tree is depicted below. In Figure 2, R_1 through

denote the rectangles in the respective nodes. R_4 through R_{10} are leaf nodes which point to data tuples or spatial objects. The rectangle R_1 covers all rectangles in its child, i.e., R_1 encloses R_4, R_5, and R_6. Similarly, R_2 is the bounded rectangle for R_7 and R_8, and R_3 for R_9 and R_{10}. Figure 3 further illustrates the intermediate and leaf rectangles as they enclose spatial objects.

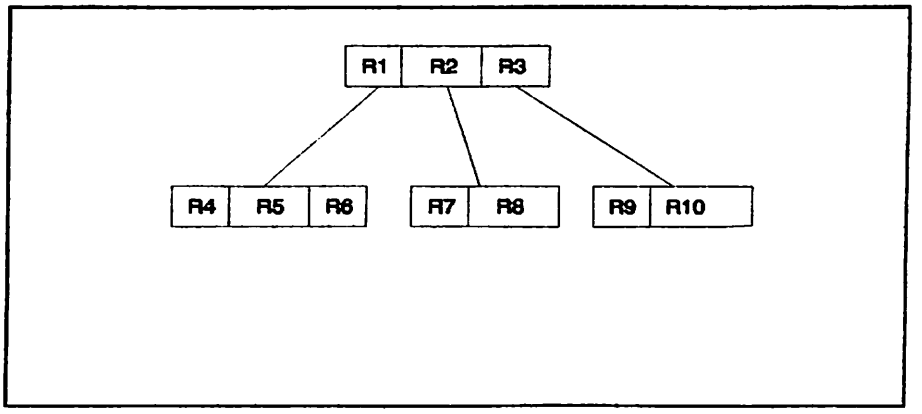

Figure 2. An R-tree

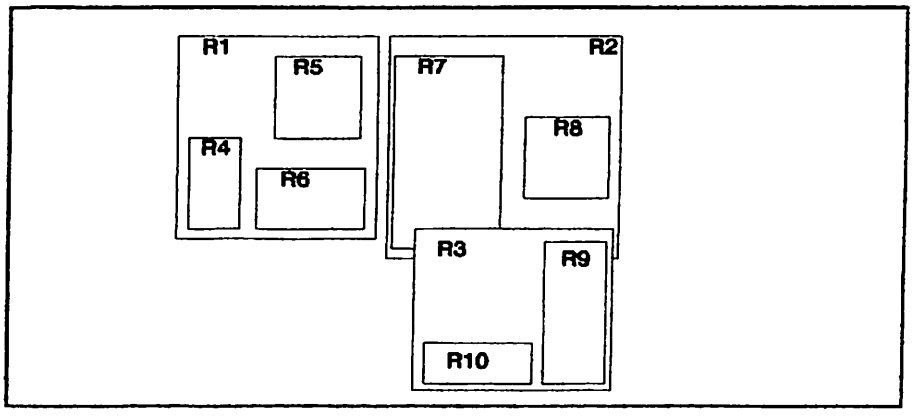

Figure 3. An example showing rectangles enclosing spatial objects for the R-tree structure given in Figure 2

Algorithms

An R-tree consists of nodes which are treated as logical pages. Logical pages are also called as node pages. There are four possible types of nodes as described in [GRE89]; a root node, a root that is also a leaf, a leaf node, and a node that is neither a root nor a leaf. The structure of a node includes the number of rectangles in the node, i.e., the number of child nodes it points to, the parent page node identifier, an array of rectangles, and an array of child page identifiers. The parent identifier is necessary for propagating splits upwards. It is also useful for implementing search and retrieval operations. The size of a page is an optimization parameter for spatial data access methods. A small page size signifies that sorting and searching individual nodes will go quickly but the tree will be deep. A large page size, on the other hand, indicates that the tree will be shallow [GRE89].

Search

The search algorithm is similar to a B-tree [GUT84]. Each node has a certain number of subtrees. Hence, more than one subtree may be needed to search for an index record. The rectangle of an index entry E is denoted as E(I) and the childpointer as E(P).

Algorithm SEARCH [GUT84]: A rectangle S is to be searched in a given R-tree with root node as H. Find all index records whose rectangle overlap S.
S1. [Search subtrees] If H is not a leaf, check each entry E to find out whether E(I) overlaps S. For all overlapping entries, call SEARCH on the R-tree whose root node is pointed to by E(P).
S2. [Search leaf node] If H is a leaf, check all entries to determine the qualifying

record which overlaps S.

Insertion

Inserting index records is also similar to a B-tree insertion routine. New index records are inserted into the leaves, overflowing nodes are split and split propagates up the tree [GUT84].

Algorithm INSERT [GUT84]: Insert a new index entry into an R-tree.
I1. [Find position for new record] Invoke CHOOSELEAF to select a leaf node L in which to place E.
I2. [Add record to leaf node] If leaf node L has vacancy to fit another entry, then insert E. Otherwise, invoke SPLITNODE to obtain L and L_2 containing E and all old entries of L.
I3. [Propagate changes upward] Call ADJUSTTREE on L, or L and L_2 if a split was performed.
I4. [Increase the height] If root node is split because of upward propagation, create a new root with its children as the two resulting nodes.

Algorithm CHOOSELEAF [GUT84]: Choose a leaf node to place the new entry E.
C1. [Initialize] Assign R as the root node.
C2. [Check if leaf node] If R is a leaf, return R.
C3. [Choose subtree] If R is not a leaf, let F be the entry in R whose rectangle F(I) needs least area enlargement to include E(I). If there is a tie, then choose rectangle with the smallest area.
C4. [Descend down to leaf] Assign R to be the child node pointed by F(P) and repeat from C2.

Algorithm ADJUSTTREE [GUT84]: Propagate node splits upward from leaf node L, and adjust covering rectangles on the path to the root.
A1. [Initialize] Set N = L. If L was split previously, set NN to be the resulting second node.
A2. [Check if complete] If N is the root node, return.
A3. [Adjust rectangles in parent entry] Let P be the parent node of N, and let E(N) be N's entry in P. Adjust this entry's I field so that it encloses all entries in N.
A4. [Propagate node split upward] Due to a split, if N has a pair N_2, create a new entry $E(N_2)$ with its child pointer P pointing to N_2 and its index entry. Add $E(N_2)$ to P if there is vacant slot. Otherwise call SPLITNODE to produce P and P_2 containing $E(N_2)$ and all old entries of P.
A5. [Climb to next level] Set N = P and set $N_2 = P_2$ if a split occurred. Repeat from A2.

Deletion

The deletion operation with regard to underfull nodes slightly differs from the corresponding operation in a B-tree [GUT84]. In a B-tree, an underfull node can be merged with the sibling that will have its area increased the least, or the orphaned entries can be distributed among sibling nodes. Reinsertion of orphaned entries was chosen by Guttman [GUT84], because the only requirement was to invoke insertion routine again. The method still accomplishes the same task and is efficient, and reinsertion refines the spatial structure of the tree and prevents slow deterioration if entries were permanently positioned under the same parent.

Algorithm DELETE [GUT84]: Delete index record E from an R-tree.
D1. [Search node containing record] Invoke FINDLEAF to find the leaf node L that contains E. Stop if the record was not found.
D2. [Delete record] Remove E from leaf node L.
D3. [Propagate changes] Call CONDENSETREE passing L.
D4. [Decrease height of tree] If the root node has only one child after the tree has been adjusted, then the child becomes the new root.

Algorithm FINDLEAF [GUT84]: Find the leaf node containing the index entry E given the R-tree and the root node, H.
F1. [Search subtrees] If H is not a leaf, check each entry F in H to find out if F(I) overlaps E(I). For each valid entry, invoke FINDLEAF on the tree whose root is pointed to by F(P). Continue until E is found or until all entries have been examined.
F2. [Search leaf node] If H is a leaf, check each entry to determine if it matches E. If E is found return H, else return indicating record was not found.

Algorithm CONDENSETREE [GUT84]: Given a leaf node L from which an entry has been removed, discard the node if it has too few entries and relocate its entries. Propagate node elimination upward as necessary. Adjust all covering rectangles on the path to the root, making them smaller if possible.
C1. [Initialize] Set N = L. Set Q, the set of eliminated nodes, to be empty.
C2. [Search for parent entry] If N is the root, go to C6. Otherwise, let P be the parent of N, and let E(N) be N's entry in P.
C3. [Eliminate underfull node] If N has fewer than m entries, delete E(N) from P

and add N to set Q.
C4. [Adjust covering rectangle] If N has not been eliminated, adjust I from E(N) to enclose all entries in N.
C5. [Climb up one level] Set N = P and repeat from C2.
C6. [Reinsert orphaned entries] Reinsert all entries of nodes in set Q by invoking the INSERT routine.

Split

Consider a node containing M, the maximum number of entries, into which a new entry needs to be added. It is necessary to partition or split the set of M+1 entries into two groups to form two new nodes.

Exhaustive Algorithm This is the straightforward method in which all possible groupings of the nodes are formed and the best one is selected with the minimum area node split. The number of cases to be searched is approximately 2^{M-1} [GUT84].

Quadratic Cost Algorithm This method tries to find the split with a small area but does not guarantee to find the smallest possible area. The algorithms are as described below [GUT84].

Algorithm QUADRATICSPLIT [GUT84]: Partition a set of M+1 entries into two groups.
Q1. [Pick first entry from each group] Invoke PICKSEEDS_Q to choose two entries to be the first elements of the groups. Assign each to a group.
Q2. [Check if complete] If all entries have been assigned, return. If one group has so few entries that all the rest must be assigned to it in order for it to have the minimum number, m, assign them and stop.
Q3. [Select entry to assign] Call PICKNEXT algorithm to choose the next entry to assign. Add it to the group whose covering rectangle will have to be enlarged least to accommodate it. Resolve ties by adding the entry to the group with smaller area, then to the one with fewer entries, then to either. Repeat from Q2.

Algorithm PICKSEEDS_Q [GUT84]: Choose two entries to be the first elements of the groups.
PQ1. [Calculate inefficiency of grouping entries together] For each pair of entries E_1 and E_2, compose a rectangle J including rectangles E_1 and E_2. Call these rectangles $E_1(I)$ and $E_2(I)$. Calculate

d = area (J) - area ($E_1(I)$) - area ($E_2(I)$).

PQ2. [Choose the most wasteful pair] Choose the pair with the largest d.

Algorithm PICKNEXT [GUT84]: Pick the next remaining entry for classifying it in one of the two groups.
P1. [Determine cost of putting each entry in each group] For each entry E not yet in a group, calculate d_1 = the area increase required in the covering rectangle of group 1 to include E(I). Calculate d_2 similarly for group 2.
P2. [Find entry with greatest preference for one group] Choose any entry with the maximum difference between d_1 and d_2.

Linear Cost Algorithm This algorithm is similar to the quadratic cost algorithm. The only distinction is that the SPLIT routine invokes PICKSEEDS_L, instead of PICKSEEDS_Q [GUT84].

Algorithm PICKSEEDS_L [GUT84]: Select two entries to be the first elements of the groups.
PL1. [Find extreme rectangles along all dimensions] Along each dimension, find the entry whose rectangle has the highest low side and the one which has the lowest high side. Record the separation.
PL2. [Adjust for shape of the rectangle cluster] Normalize the separations by dividing by the width of the entire set along the corresponding dimension.
PL3. [Select the most extreme pair] Choose the pair with the greatest normalized separation along any dimension.

Program Execution

During the initial stages of the implementation process, an R-tree was developed. The properties of the R-tree as given in [GUT84] were followed as part of the requirements. The rectangles or index records were generated using a random number generator as given in [PAR88]. A covering parameter was used to set the

limits on the value of each coordinate, i.e., xlow, ylow, length of x, and length of y. Covering parameter is defined as the ratio of the sum of the areas of the generated rectangles to the area of the total bounding region.

The program executes by initially asking the user to select one of the following:

1. Random Program Execution
2. Exit

Upon selection of the first choice, the program requests the user to create a header name for the R-tree and to enter the boundaries for the coordinates that will be generated.

Enter id of R-tree:

Enter the maximum bounding coordinates(minx, miny, maxx, maxy):

Next, the covering parameter is requested to confirm that the rectangles generated obey the limits. Otherwise, the values are adjusted before the rectangles are inserted.

Enter the covering parameter (> 0.01):

After the initial conditions are met, the coordinates for the rectangles are generated. Each one of these index records are inserted into the R-tree. Additionally, to prove the validity of other basic operations like access or search for an index record, deletion, breadth-first traversal and depth-first traversal, and elimination of R-tree, certain rectangles are chosen and the tree is traversed. Some of the records are searched to confirm the search routine. The statistics of the R-tree is furnished after insertion and deletion. An example of the parameters for a sample observation is given below.

Maximum number of rectangles in a node (M)	=	5
Minimum number of rectangles in a node (m)	=	2
Covering parameter (assigned)	=	0.25
Covering parameter (actual)	=	0.25
Height of the R-tree	=	2
Number of rectangles	=	9
Number of nodes	=	4
Number of leaf nodes	=	3

Thus the R-tree structure is observed to be useful for non-zero size spatial objects. It would work especially well in conjunction with abstract data types and abstract indexes to streamline the handling of spatial data [GUT84].

CHAPTER III

IMPLEMENTATION

Initially, a database was set up by using VLSI data as an application. Two different sets of VLSI data were used to validate the system. This database was extracted from a VLSI design layout. Rectangles denoting regions in the VLSI layout were generated. Hence, retrieval into these regions could be done using a spatial database management system. The system that was developed to handle these spatial objects was an index structure that is pertinent to the family of R-trees. It was called an R^*-tree [BEC90] because certain modifications were made to the algorithms for the split and reinsert operations. The operations that could be performed on the R^*-tree were insertion, deletion, search, breadth-first traversal, and depth-first traversal. A menu-driven interface was implemented to aid novice users in performing various operations easily. The access operation on any given search area displayed all overlapping rectangles that overlap within the area, and the exact match rectangle if the query was for determining a particular object. Some statistics were gathered on the system by varying the maximum number of rectangles that can occupy a node. A description of the index structure and the operations associated with it is contained in the following section.

R*-tree

An R*-tree [BEC90] is a data structure that can handle multidimensional point and spatial data. An R*-tree has the same node structure as an R-tree. The spatial objects are stored in leaf level nodes. The interior nodes consist of rectangles that encloses all its children and rectangles at lower levels. The key to the robustness of this index structure is based on the values of the area, margin, and overlap of the directory rectangles. Since all three parameters are reduced, ugly data distributions are prevented. Furthermore, due to the concept of forced reinsert after deletion, splits can be avoided, the structure is dynamically reorganized, and its storage utilization is higher than that for other R-tree variants [BEC90].

An R*-tree is an enhanced variant of the R-tree. It is also similar to a B-tree and has a uniform height tree structure. It stores multidimensional rectangles as complete objects without clipping or transforming them into higher dimensional points.

The R*-tree has the following structure [BEC90]. A non-leaf node contains entries of the following format

(cp, Rectangle)

where cp is a pointer to a child node in the R*-tree, and Rectangle is the minimum bounding rectangle of all rectangles which are entries in that child node. Leaf nodes contain the following pair of entries.

(Oid, Rectangle)

where Oid refers to a object identifier or a record in the spatial object database, and Rectangle is the enclosing rectangle for that spatial object.

An example of an R*-tree is shown below. The two figures show the difference in the splits of the R-tree (m = 30%) and the R*-tree (m = 40 %) [BEC90].

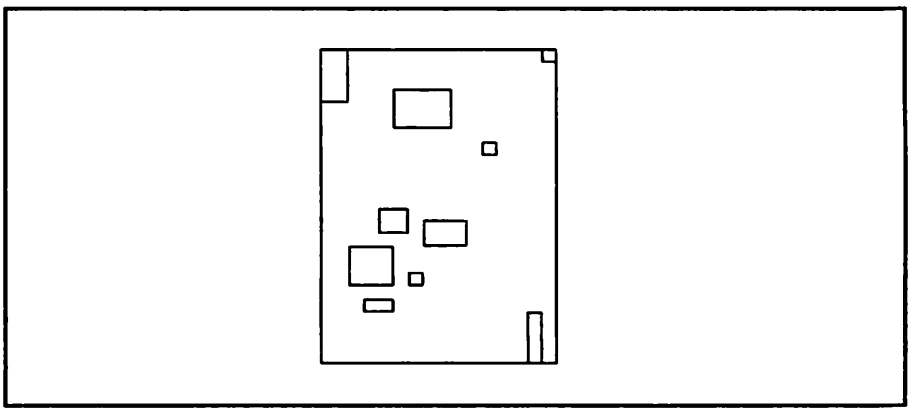

Figure 4. An overfull node (M = 9) (Source: [BEC90])

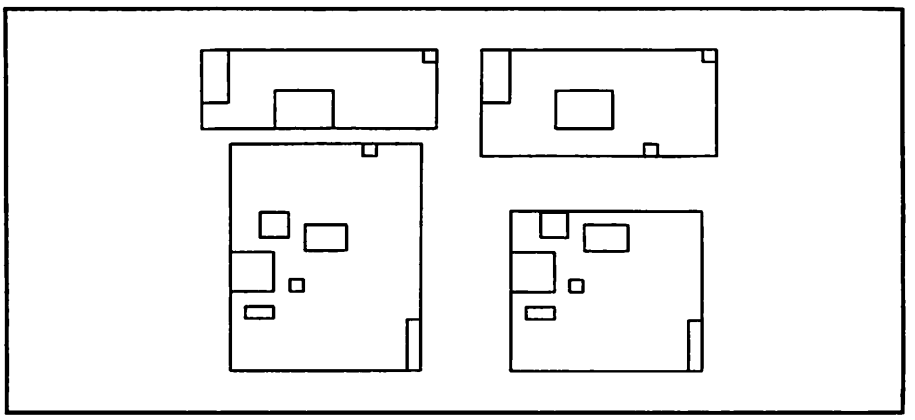

Figure 5. Split in an R-tree and R*-tree (Source: [BEC90])

In the above figures, an overfull node is given in which it is assumed that the

maximum rectangles per node is 9. The minimum number of rectangles can be any value between 2 and 5. It is observed that the split of an R*-tree (m = 4) is more uniform than the split of an R-tree (m = 3).

Properties

Let M be the maximum number of entries that can be accommodated in one node, and m be the minimum number of entries ($2 \leq m \leq M/2$). The R*-tree satisfies the following properties [BEC90]:

(1) The root has at least two children unless it is a leaf.

(2) All leaves appear on the same level.

(3) Every leaf node has between m and M children unless it is the root.

(4) Every non-leaf node has between m and M entries unless it is the root.

The R*-tree is also dynamic, just like an R-tree. Insertions and deletions can be intermingled with queries such as accessing with no necessity for periodic global reorganization. Thus, the structure allows for overlapping directory rectangles. Hence, it cannot be guaranteed that only one search path is required for a query involving an exact match [BEC90, GUT84].

In an R-tree, certain parameters for obtaining good retrieval are interdependent. Some of the parameters as discussed by Beckmann et al. [BEC90] are as follows:

(1) Minimization in the area covered by a bounding rectangle.

This implies that the area of the bounding rectangle, i.e., the rectangles that

are formed in the interior nodes of the R*-tree, should be reduced to a minimum to improve performance. Thus, traversal decisions for exact matches can be taken at a higher level in the tree.

(2) Minimization in the overlap between directory rectangles

If there is less overlap between bounding rectangles, then the number of paths to be traversed also decreases.

(3) Minimization in the margin of a directory rectangle.

The margin is the sum of the lengths of the edges, or the perimeter of a rectangle. By minimizing the margin, the rectangles will get clustered into bounding boxes with only little variance in the lengths of the edges. This will eventually reduce the area of directory rectangles.

(4) Storage utilization should be optimized to utilize only the required space in memory. If the area and overlap of a directory rectangle are minimized, the storage utilization will be low. By minimizing the margins, storage utilization can be reduced. Queries with large query rectangles will be affected more by the storage utilization than by the other parameters [BEC90].

The area of a rectangle is nothing but the multiplication of the length and width of a rectangle. The margin is the sum of the lengths of edges or the perimeter of a rectangle. The overlap of an entry in a node is defined as follows [BEC90]: Let E_1, E_2, ..., E_m be the entries in a node, then

$$\text{overlap } (E_j) = \sum_{\substack{k = 1 \\ k \neq j}}^{p} \text{area } (E_j.\text{Rectangle} \cap E_k.\text{Rectangle})$$

where $1 \leq j \leq p$.

Algorithms

The various algorithms used to implement the operations to be performed on an R*-tree are given below.

Search

The search algorithm is the same as described earlier in the case of an R-tree. It uses a similar principle of searching as in a B-tree [GUT84].

Algorithm RS_SEARCH [GUT84]:
S1. [Initialize] Set H = root node.
S2. [Search subtrees] If H is not a leaf, check each entry E to determine whether E[i] overlaps search rectangle S. For all overlapping entries, call RS_SEARCH on the R*-tree whose root node is pointed to by E[cp].
S3. [Search leaf node] If H is a leaf, check all entries to determine the qualifying record which overlaps S.

Insertion

Beckmann et al. described the insertion routine which provides the best retrieval performance [BEC90]. The insertion algorithm chooses the appropriate insertion path based on the area and overlap parameters. In the RS_CHOOSESUBTREE algorithm, the nearly minimum overlap cost is determined in which the entries are sorted by increasing area size before minimum overlap is found.

Algorithm RS_CHOOSESUBTREE [BEC90]:
C1. [Initialize] Set N = root.
C2. [Check for leaf]

If N = leaf
    return N
else
    if the childpointers in N point to leaves
    [determine the nearly minimum overlap cost]
        Sort the rectangles in N in increasing order of their area enlargement needed to include the new data rectangle. Let A be the group of first p entries. From the entries in A, considering all entries in N, choose the entry whose rectangle needs least overlap enlargement. Resolve ties by choosing the entry whose rectangle needs least area enlargement.
    then
        the entry with the rectangle of smallest area.
    if the childpointers in N do not point to leaves
    [determine the minimum area cost]
        Choose the entry in N whose rectangle needs least area enlargement to include the new data rectangle. Resolve ties by choosing the entry with the rectangle of smallest area.
C3. [Continue] Set N = child node pointed to by the childpointer of the chosen entry and
repeat from C2.

The value of p, i.e., the first p entries after sorting the rectangles based on their area, is set to 32 based on experiments for two-dimensional objects [BEC90]. The VLSI data application is also two dimensional and hence this value was taken for building the R*-tree index structure.

Split

The splitting algorithm utilizes the area, margin, and overlap parameters to find good splits. The following method was used [BEC90].

Along the x-axis and the y-axis, the rectangles in a node to be split are sorted by their lower value of x and y coordinates and then by the upper value of the x and y coordinates. Let M be the maximum number of entries in a node and m be the

minimum number of entries, then, for each of the four sorts, M-2m+2 distributions of the M+1 entries into two groups are determined. The k^{th} distribution, where k = 1, ..., (M-2m+2), is described as follows. The first group contains the first (m-1)+k entries and the second group contains the remaining entries. Three different goodness values are used for determining the final distribution. They are as follows.

(1) area-value = area [bb(first group)] + area [bb(second group)]

(2) margin-value = margin [bb(first group)] + margin [bb(second group)]

(3) overlap-value = area [bb(first group) $\cap$ bb(second group)]

where bb denotes the bounding box of a set of rectangles.

The split algorithm is described as follows [BEC90].

Algorithm RS_SPLIT [BEC90]:
S1. Call RS_CHOOSESPLITAXIS to determine the axis perpendicular to which the split is performed.
S2. Call RS_CHOOSESPLITINDEX to determine the best distribution into two groups along that axis.
S3. Distribute the entries into two groups.

Algorithm RS_CHOOSESPLITAXIS [BEC90]:
A1. For each axis
Sort the entries by the lower x and y values, then by the upper x and y values of their rectangles, and determine all distributions as described above. Compute S, the sum of all margin values of the different distributions.
A2. Choose the axis with the minimum S as the split axis.

Algorithm RS_CHOOSESPLITINDEX [BEC90]:
I1. Along the chosen split axis, select the distribution with the minimum overlap value. Resolve ties by choosing the distribution with minimum area value.

The experiments conducted by Beckmann et al. [BEC90] showed that the split algorithm works well for m = 40%.

Deletion

The insertion of rectangles into the R*-tree or the R-tree is non-deterministic, i.e., various sequences of insertions can lead to different tree structures. In such cases, the directory rectangles that are already formed cannot guarantee good retrieval performance and hence a reorganization is required to make it a powerful index structure.

After a deletion operation in an R*-tree, the orphaned entries are reinserted during insertion routine on every level of the tree [BEC90, GUT84]. The complete deletion algorithm, including the initial procedures to delete, which is the same as the R-tree, is described below.

Algorithm RS_DELETE [BEC90]:
D1. [Search node containing record] Invoke RS_FINDLEAF to find the leaf node L that contains E. Stop if the record was not found.
D2. [Delete record] Remove E from leaf node L.
D3. [Propagate changes] Call RS_CONDENSETREE passing L.
D4. [Decrease height of tree] If the root node has only one child after the tree has been adjusted, then the child becomes the new root.

Algorithm RS_FINDLEAF [BEC90]:
F1. [Search subtrees] If H is not a leaf, check each entry F in H to find out if F[I] overlaps E[I]. For each valid entry, invoke RS_FINDLEAF on the tree whose root is pointed to by F[P]. Continue until E is found or until all entries have been examined.
F2. [Search leaf node] If H is a leaf, check each entry to determine if it matches E. If E is found return H, else return indicating record was not found.

Algorithm RS_CONDENSETREE [BEC90]:
C1. [Initialize] Set N = L. Set Q, the set of eliminated nodes, to be empty.
C2. [Search for parent entry] If N is the root, go to C6. Otherwise, let P be the parent of N, and let E[N] be N's entry in P.
C3. [Eliminate underfull node] If N has fewer than m entries, delete E[N] from P and add N to set Q.
C4. [Adjust covering rectangle] If N has not been eliminated, adjust I from E[N] to enclose all entries in N.

C5. [Climb up one level] Set N = P and repeat from C2.
C6. [Reinsert orphaned entries] Reinsert all entries of nodes in set Q by invoking the RS_INSERTDATA routine.

Algorithm RS_INSERTDATA [BEC90]:
ID1. Invoke RS_INSERT starting with the leaf level as a parameter, to insert a new data rectangle.

Algorithm RS_INSERT [BEC90]:
I1. Invoke RS_CHOOSESUBTREE with the level as a parameter, to find an appropriate node N, in which to place the new entry E.
I2. If N has less than M entries, accommodate E in N. If N has M entries, invoke RS_OVERFLOWTREATMENT with the level of N as a parameter (for reinsertion or split).
I3. If RS_OVERFLOWTREATMENT was called and a split was performed, propagate overflow treatment upwards if necessary. If overflow treatment caused a split of the root, create a new root.
I4. Adjust all covering rectangles in the insertion path such that they are minimum bounding boxes enclosing their children rectangles.

Algorithm RS_OVERFLOWTREATMENT [BEC90]:
O1. If the level is not the root level, and this is the first call of overflow treatment in the given level during the insertion of one data rectangle, then
        invoke RS_REINSERT
else
        invoke RS_SPLIT.

Algorithm RS_REINSERT [BEC90]:
RI1. For all M+1 entries of a node N, compute the distance between the centers of their rectangles and the center of the bounding rectangle of N.
RI2. Sort the entries in decreasing order of their distances computed in RI1.
RI3. Remove the first p entries from N and adjust the bounding rectangle of N.
RI4. In the sort defined in RI2, starting with the minimum distance (closed reinsert), invoke RS_INSERT to reinsert the entries.

Various values were tested experimentally [BEC90] and the one that yields the best performance is p = 30% of M for the leaf as well as non-leaf nodes. Closed reinsert, as opposed to far reinsert where maximum distance is used, reduces the size of the bounded rectangle.

Thus, forced reinsert decreases the overlap and as a side effect, storage

utilization is improved. As a result of restructuring, less splits occur. The cpu cost is high since insert is invoked often, but lesser number of splits have to be performed.

Program Execution

An R*-tree was developed by making certain modifications to the R-tree. The properties of the R*-tree as given by Beckmann et al. were followed [BEC90]. The rectangles or index records were initially generated using a random number generator as given by Park and Miller [PAR88]. A covering parameter was used to set the limits on the value of each coordinate, i.e., xlow, ylow, length of x, and length of y. The operations that were used to test the validity of the R*-tree were similar to that performed in the R-tree. A menu driven interface was used with interaction from the user regarding various parameters. A header identifier was requested along with the limits on the bounding region. The maximum limit on the coordinates was used to guard the random number generator from generating rectangles outside the boundaries. Moreover, the covering parameter as specified by the user ensured that the area of the generated rectangle was less than the area of the maximum bounding region. The rectangles which conformed to the specifications were inserted into the R*-tree. A breadth-first traversal of the tree was done to confirm the validity of the insertion. The R*-tree was traversed using the depth-first method too. A certain set of rectangles were chosen to be searched to confirm that the search routine worked successfully. The statistics of the R*-tree was taken. Next, all the rectangles were deleted from the R*-tree and a breadth-first traversal was performed. All the

rectangles including the header were deleted from the R*-tree.

A VLSI design layout file was taken as an application that was used to validate the system. The VLSI data file was first converted into a text file with many rectangles being generated. This text file was used to confirm the various operations.

The program executes by initially asking the user to select one of the following options.

1. Run Interactive Program
2. Exit

Upon selection of the first choice, another menu was displayed which allowed the user to choose any of the following options.

R*-tree

1. Create Header
2. Insert Rectangle
3. Delete Rectangle
4. Access Leaf Record
5. Breadth First Traversal
6. Depth First Traversal
7. Statistics
8. Kill R*-tree

Enter your choice :

For the first choice, the program requests the user to create a header name for the R*-tree.

Enter header id of R*-tree:

The program allowed the user to create a forest of R*-trees if necessary. Hence, different headers with unique identifiers for the R*-tree could be created. Next, by selecting number 2 in the second menu, the user was requested to confirm the

insertion of all rectangles from the VLSI data file.

Do you wish to insert all rectangles given in vlsidata1 (y/n) :

Some of the rectangles in the VLSI data application were duplicates, or had the same dimensions. Those rectangles were inserted only once. Next, the user had the option of accessing or searching for a leaf record, deleting a record, breadth first traversal or depth first traversal, obtain statistics or terminate the R*-tree by deleting the nodes and the header. In the case of searching for a spatial object (selection 4), the rectangles that the leaf level pointed to were searched for a match. The user was asked to give the header id of the tree to be searched and the coordinates of the rectangle.

Enter coordinates of rectangle to be searched (xlow, ylow, length_x, length_y):

An appropriate message was given based on whether the record was found or not. Selection 3 allowed the user to delete a record or rectangle from the R*-tree. Again, the header id was given as well as the coordinates. The changes were reflected in the statistics.

Enter the rectangle to be deleted (xlow, ylow, length_x, length_y):

Choices 5 and 6 allowed the user to visualize the rectangles created or modified, after insertion or deletion, by traversing breadth-first or depth-first.

The statistics of the R*-tree (number 7) was furnished upon giving the appropriate header id. An example of the parameters for a sample observation were as follows

Maximum number of rectangles in a node (M) = 5

Minimum number of rectangles in a node (m) = 2

Covering parameter (assigned)	=	0.00
Covering parameter (actual)	=	0.00
Height of the R-tree	=	6
Number of rectangles	=	1464
Number of nodes	=	606
Number of leaf nodes	=	426

Finally, choice number 8 allowed the user to delete the header of any R*-tree given the header identifier. This lead the user back to the main menu with a choice of continuing the program or terminating it.

Thus the R*-tree structure was found to be robust with the introduction of new concepts like area, margin, and overlap of directory rectangles. The number of splits was reduced due to forced reinsert routine, the structure was dynamically reorganized, and the storage utilization was slightly high. The average insertion cost of the R*-tree was lower than for the R-tree [BEC90].

Program Environment

The environment that was used to develop the program was a Sequent Symmetry S-81 machine. The operating system used on this machine was DYNIX/ptx, which is a version of UNIX. The program was written using the C programming language. The ANSI C compiler was used to compile and execute the program. The performance of the system was observed for the R-tree and the R*-tree index structures and statistics were obtained.

Application

The application with which the spatial database system was tested consisted of a VLSI data set where numerous rectangle coordinates were given. These coordinates were obtained directly from an VLSI design layout. The layout was divided into sections or grids so that rectangles that were generated were treated as spatial objects. The rectangular coordinates were stored in the leaf levels of the index structure. A sample set of the VLSI design layout, that was converted into text file for easier storage into the database, is given in Appendix B. The grids formed from the VLSI data file were numerous rectangles which correspond to actual spatial objects.

Two VLSI data sets were used to validate the performance of the program. The first VLSI data file consisted of 1937 rectangles. Of these, only 1464 rectangles were unique. The remaining 473 rectangles were duplicates. The program checked for duplicate rectangles before inserting them into the R^*-tree. The second VLSI data file contained 4085 rectangles, of which 3047 rectangles were unique and the rest were duplicates. Those duplicates were not inserted into the index structure.

CHAPTER IV

RESULTS

A spatial database system was developed using the R*-tree as the spatial index structure. The R*-tree index structure was built using the C programming language. The database that was used to test the program was by randomly generating rectangles enclosing fictitious spatial objects. The random number generator that was used to generate coordinates for the rectangles is described by Park and Miller [PAR88].

The program was tested by first generating a certain number of rectangles based on the maximum bounding regions for the coordinates along the x-axis and the y-axis, and on the covering parameter. The covering parameter is defined as the ratio of the sum of the areas of rectangles in an R*-tree to the area of the maximum bounding region of the tree. The covering parameter was used to determine the number of rectangles that can be generated at a time. Each rectangle was inserted into the R*-tree. Next, to verify that the insertion was correctly performed, two types of traversals on the R*-tree index structure were done. The breadth-first traversal was done first followed by the depth-first traversal. To confirm that the search operation worked successfully, some of the inserted rectangles were accessed. After the

accessing of rectangles was complete, a statistical analysis was taken for the R*-tree. The values of M, the maximum size of each node in the tree, and m, the minimum limit on the number of rectangles in the node were shown. Moreover, the actual and assigned covering parameters were stated and so were the height of the tree, the number of rectangles contained in the tree and the number of leaf nodes contained in the R*-tree. Finally, to ensure the verification of deletion operation, all the rectangles that were inserted through the insert routine were deleted, until the R*-tree became empty. Another breadth-first traversal operation was performed to confirm the deletion of the tree. This process was repeated for different parameters to test the validity of the index structure.

The program was then modified to suit a VLSI data application. An example of a sample VLSI data set is given in Appendix B. The method by which the input data set was changed such that actual rectangles enclosed real spatial objects, was used to analyze the performance of the R*-tree index structure in an actual application. The index structure that was used in this study was an improvement over the other family of R-tree structures. Hence better retrieval, storage utilization, and node utilization were obtained. All the factors concerned with interdependencies among various parameters like covering parameter, M, and m, the maximum and minimum limit on the bounding region for generation of random rectangles, etc. were taken to make the index structure efficient. Some of the differences noted between the structure obtained with an R-tree and with an R*-tree show marked changes in the number of rectangles in the R*-tree, the number of leaf nodes, the height of the

tree and a very small difference in the actual and assigned covering parameters for the same limits of the maximum and the minimum size of each node in the tree.

It can be observed from the following graphs that the R*-tree utilized less memory space than the R-tree. The internal nodes and the leaf nodes in an R*-tree were comparatively fewer than the corresponding nodes for an R-tree for different values of M (the maximum number of rectangles in a node). The upper limit on the value of m (the minimum number of rectangles) was taken, i.e., M/2. The node utilization was calculated as follows.

$$\text{Node utilization} = \frac{\text{Total number of rectangles in the R*-tree}}{\text{Total number of nodes} * M}$$

It was observed that the node utilization is higher in an R*-tree compared to an R-tree. This is true for the two sets of VLSI data. The graphs below show the number of interior nodes and leaf nodes allocated in the R*-tree for VLSI data set 1 and VLSI data set 2, respectively. The graphs show the comparison between the R*-tree (represented by solid line) and the R-tree (represented by thin line). The comparison shows a small but significant decrease in storage utilization.

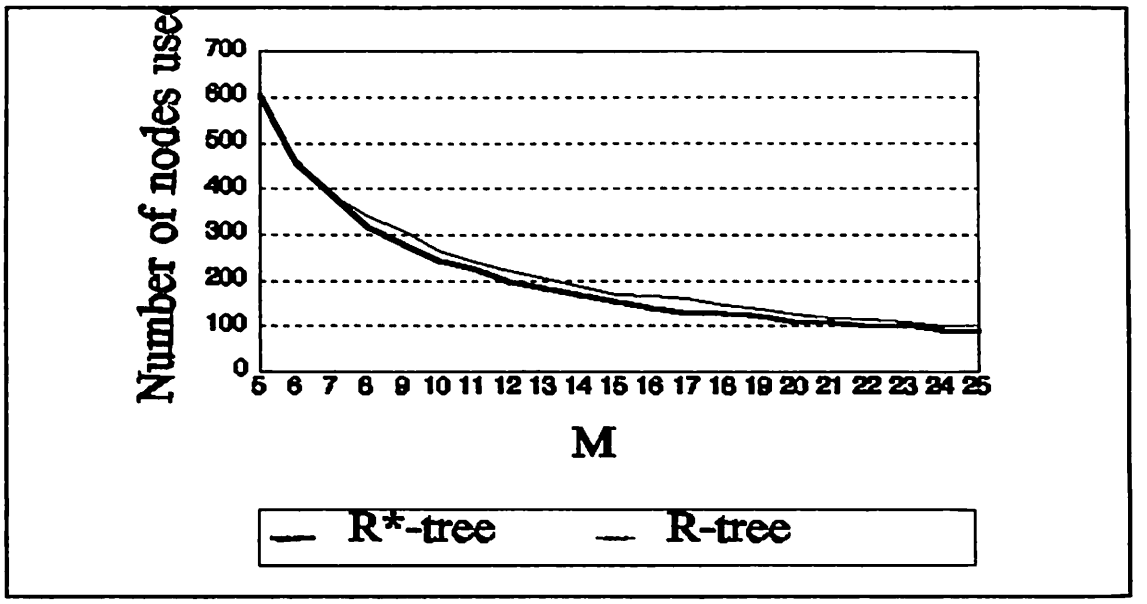

Figure 6. Number of nodes used in an R*-tree and an R-tree using VLSI data set 1

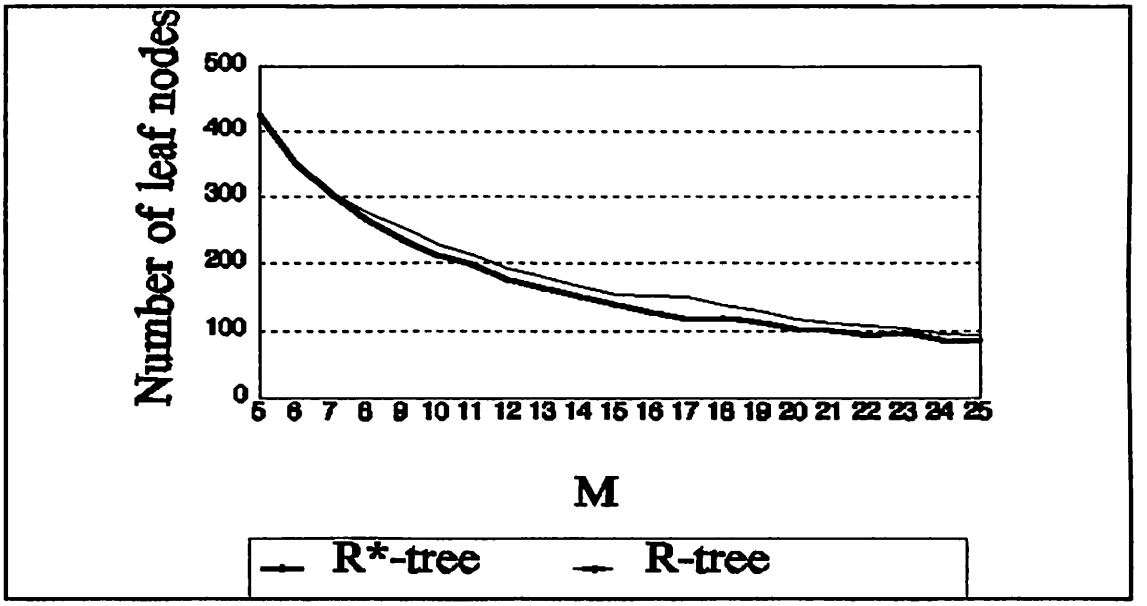

Figure 7. Number of leaf nodes used in an R*-tree and an R-tree using VLSI data set 1

The two figures below illustrate the same information using VLSI data set 2.

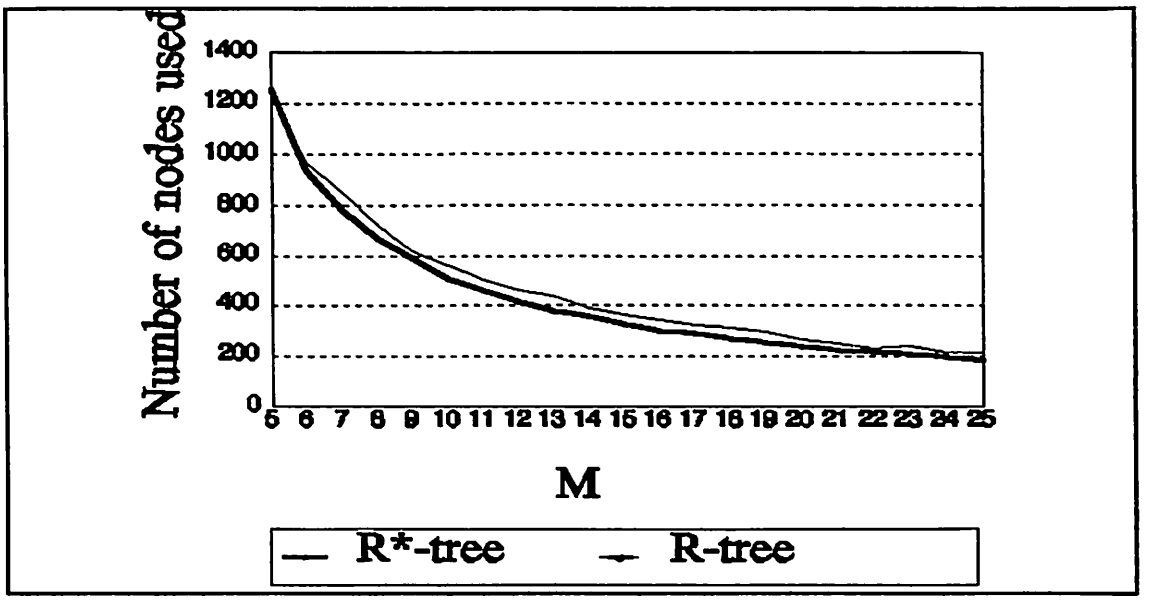

Figure 8. Number of nodes used in an R*-tree and an R-tree using VLSI data set 2

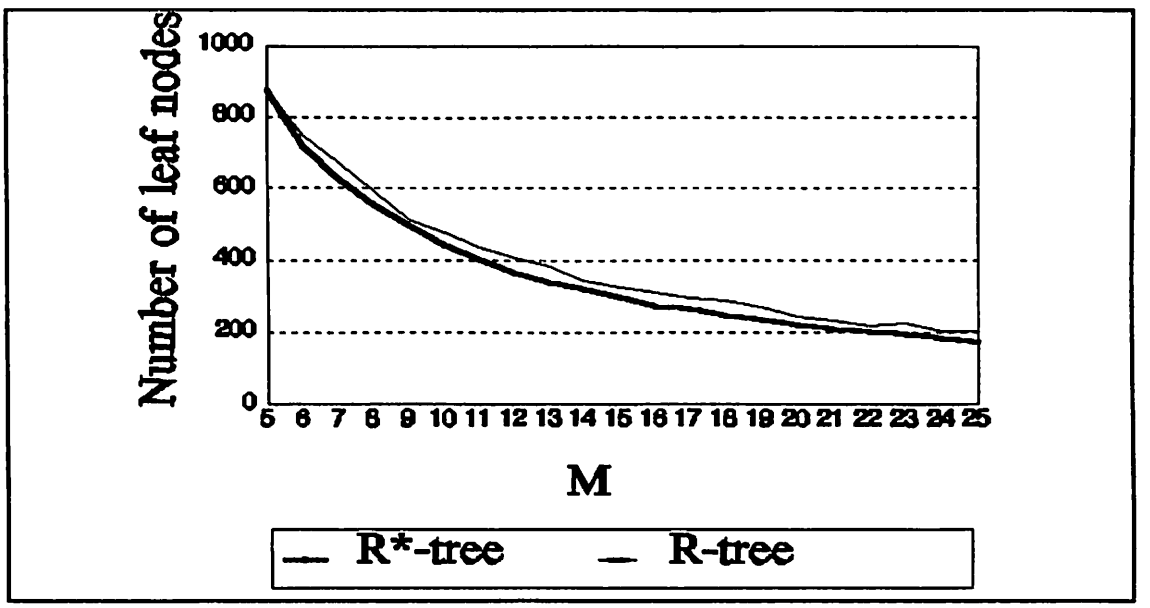

Figure 9. Number of leaf nodes used in an R*-tree and an R-tree using VLSI data set 2

The following set of graphs show marked differences in the node utilization in an R-tree and R*-tree. The calculation was done as mentioned earlier. The results

signify that the R*-tree is a better index structure than an R-tree.

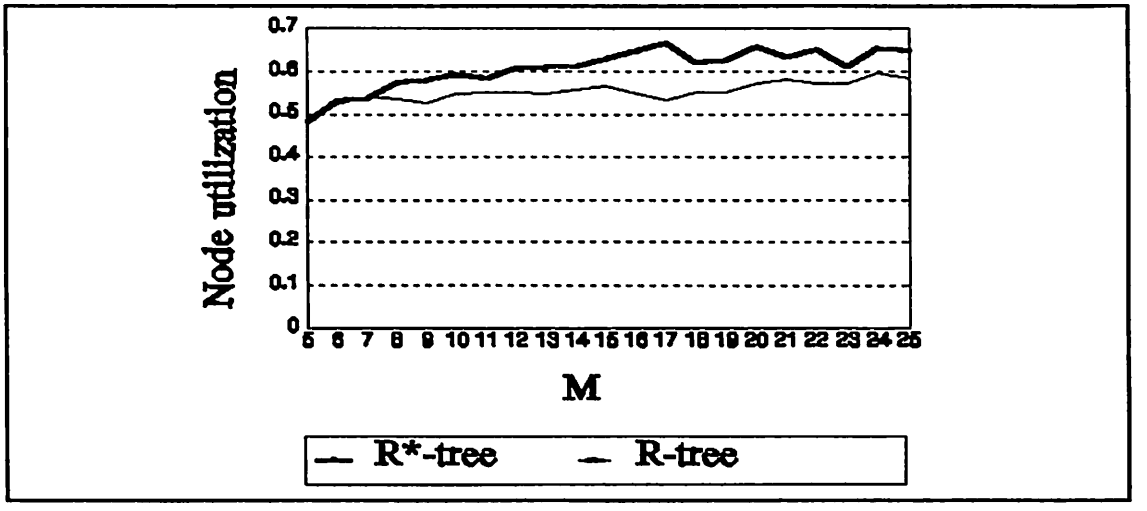

Figure 10. Node utilization for an R*-tree and an R-tree using VLSI data set 1

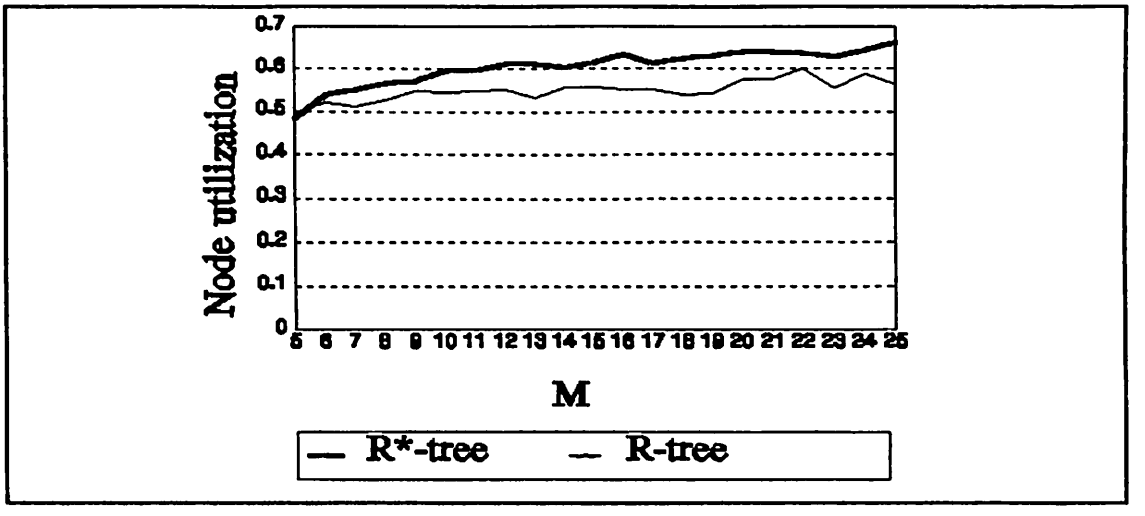

Figure 11. Node utilization for an R*-tree and an R-tree using VLSI data set 2

The results of the spatial database system clearly show that prototypes of this kind are useful for applications involving very large databases such as spatial

applications. Geographic systems, computer-aided design and geometric applications would all benefit from such systems with enhanced facilities like querying and excellent user interfaces.

CHAPTER V

SUMMARY AND CONCLUSION

A spatial database system was built using the R*-tree spatial index structure. The database that was built to validate the system was a VLSI design layout application consisting of rectangles that enclose spatial objects. Various index structures for retrieval of large spatial objects were introduced and a survey on recent literature was done on different types of index structures. The R-tree was taken into consideration for retrieval of point and spatial data. Using the R-tree suffers from certain drawbacks as mentioned in Chapter 2. Hence the R*-tree index structure was used since various parameters were taken into account for better performance on the spatial data as regards insertion, deletion, and access.

A spatial database system using the R*-tree as the index structure was described. A review of literature from recent articles reveals improvements in the family of R-trees, and one of them consists of the R*-tree. Hence, a better index structure was used to check for its performance in a spatial index system.

A number of observations were made on the performance of the system using two sets of VLSI data. The graphs were generated for different values of the maximum number of rectangles that can be accommodated in a node. The

observations from the graphs revealed that the R*-tree performs better than the R-tree in terms of storage utilization and node utilization. Moreover, searching and retrieval were performed after the user specified the position of the coordinates. All rectangles that overlap with the given search area were retrieved. A query involving an exact match rectangle resulted in retrieval of the actual leaf record. One of the parameters used for good split operations in the R*-tree was the minimization of overlap in a node. Hence, the search operation ensured that there was at most one path from the root of the R*-tree to the rectangle that is to be retrieved. But, in cases where, even after minimization, the overlap was inevitable, there could be more than one path to be searched from the root of the R*-tree down to the leaf node that points to the spatial object.

This prototype can be used for various geographic information systems, very large scale computer aided design, and various applications involving large databases. Future research and improvement could introduce queries into this database which would enhance the interface between the user and the database and make the operations simpler to perform. Further, using graphical user interfaces, a visual structure of the tree can be displayed and every operation can be clearly chalked out without manual effort. Hence, very large database applications can benefit from such systems where retrieval and other operations on such huge data can be easily performed.

REFERENCES

[AHO74]
A.V. Aho, J.E. Hopcroft, and J.D. Ullman, The Design and Analysis of Computer Algorithms, Addison Wesley, Reading, Massachusetts, 1974.

[BEC90]
Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger, "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles", Proceedings of the ACM-SIGMOD International Conference on the Management of Data, pp 322-331, Atlantic City, New Jersey, May 1990.

[BEN75]
J.L. Bentley, "Multidimensional Binary Search Trees Used for Associative Searching", Communications of the ACM, Vol. 18, 9, (September 1975), pp 509-517.

[BEN79]
J.L. Bentley and J.H. Friedman, "Data Structures for Range Searching", ACM Computing Surveys, Vol. 11, 4, (December 1979), pp 397-409.

[BEN91]
Dan Benson and Greg Zick, "Symbolic and Spatial Database for Structural Biology", Proceedings of OOPSLA 1991, pp 329-339, Phoenix, Arizona, October 1991.

[BLA90]
H.M. Blanken, A. Ijbema, P. Meek, and B. van den Akker, "The Generalized Grid File: Description and Performance Aspects", Proceedings of the IEEE Sixth International Conference on Data Engineering, pp 380-388, Los Angeles, California, February 1990.

[DAN85]
S.P. Dandamudi and P.G. Sorenson, "An Empirical Performance Comparison of Some Variations of the K-d tree and BD-tree", International Journal of Computer and Information Sciences, Vol. 14, 3, (June 1985), pp 135-159.

[DAN86]
S.P. Dandamudi and P.G. Sorenson, "Algorithms for BD-trees", Software - Practice

and Experience, Vol. 16, 12, (December 1986), pp 1077-1096.

[DAT92]
C.J. Date, An Introduction to Database Systems, Vol. 1, 5th Edition, Addison Wesley Publishers, 1992.

[FAL87]
C. Faloutsos, T. Sellis, and N. Roussopoulos, "Analysis of Object-Oriented Spatial Access Methods", Proceedings of the ACM-SIGMOD International Conference on the Management of Data, pp 426-439, San Francisco, California, December 1987.

[FIN74]
R.A. Finkel and J.L. Bentley, "Quad Trees - A Data Structure for Retrieval on Composite Keys", Acta Informatica, Vol. 4, 1974, pp 1-9.

[FRE87]
M. Freeston, "The BANG File: A New Kind of Grid File", Proceedings of the ACM-SIGMOD International Conference on the Management of Data, pp 260-269, San Francisco, December 1987.

[GAR82]
I. Gargantini, "An Effective Way to Represent Quad Trees", Communications of the ACM, Vol. 25, 12, (December 1982), pp 905-910.

[GRE89]
D. Greene, "An Implementation and Performance Analysis of Spatial Data Access Methods", Proceedings of the IEEE Fifth International Conference on Data Engineering, pp 606-615, Los Angeles, California, February 1989.

[GUE90]
O. Guenther and A. Buchmann, "Research Issues in Spatial Databases", ACM-SIGMOD Record, Vol. 19, 4, (December 1990), pp 61-68.

[GUN89]
O. Gunther, "The Design of the Cell Tree: An Object-Oriented Index Structure for Geometric Databases", Proceedings of the IEEE Fifth International Conference on Data Engineering, pp 598-605, Los Angeles, California, February 1989.

[GUT84]
Antonin Guttman, "R-trees: A Dynamic Index Structure for Spatial Searching", Proceedings of the ACM-SIGMOD International Conference on the Management of Data, pp 47-57, Boston, Massachusetts, June 1984.

[HUT90]
A. Hutflesz, H-W. Six, and P. Widmayer, "The R-File: An Efficient Access Structure for Proximity Queries", Proceedings of the IEEE Sixth International Conference on Data Engineering, pp 372-379, Los Angeles, California, February 1990.

[KNU68]
D. Knuth, The Art of Computer Programming, Vol. I, Addison-Wesley, Reading, Massachusetts, 1968.

[NIE84]
J. Nievergelt, H. Hinterberger, and K.C. Sevcik, "The Grid File: an Adaptable, Symmetric Multikey File Structure", ACM Transactions on Database Systems, Vol. 9, 1, pp 38-71, 1984.

[NIE89]
J. Nievergelt, "7 +- 2 criteria for assessing and comparing spatial data structures", Proceedings SSD1989, Lecture Notes in Computer Science, No. 409, Springer Verlag, 1990.

[OHS83]
Y. Ohsawa and M. Sakauchi, "BD-tree: A New N-dimensional Data Structure with Efficient Dynamic Characteristics", Proceedings of the IFIP 9th World Computer Congress, pp 539-544, Paris, France, September 1983.

[OHS90]
Y. Ohsawa and M. Sakauchi, "A New Tree Type Data Structure with Homogenous Nodes Suitable for a Very Large Spatial Database", Proceedings of the IEEE Sixth International Conference on Data Engineering, pp 296-303, Los Angeles, California, February 1990.

[PAR88]
Stephen K. Park and Keith W. Miller, "Random Number Generators: Good Ones are Hard to Find", Communications of the ACM, Vol. 31, 2, (October 1988), pp 1192-1201.

[ROB81]
J.T. Robinson, "The K-d-B-tree: A Search Structure for Large Multidimensional Dynamic Indexes", Proceedings of the ACM-SIGMOD International Conference on the Management of Data, pp 10-18, April 1981.

[SAM84]
Hanan Samet, "The Quadtree and Related Hierarchical Data Structures", ACM Computing Surveys, Vol. 16, 2, (June 1984), pp 187-260.

[SAM89]
Hanan Samet, The Design and Analysis of Spatial Data Structures, Addison Wesley, Reading, Massachusetts, 1989.

[SAM90]
Hanan Samet, Applications of Spatial Data Structures: Computer Graphics, Image Processing and GIS, Addison Wesley, Reading, Massachusetts, 1990.

[SEE90]
B. Seeger and H.P. Kriegel, "The Buddy-Tree: An Efficient and Robust Access Method for Spatial Database Systems", Proceedings of the Conference on VLDB, pp 590-601, 1990.

[SEL87]
Timos Sellis, Nick Roussopoulos, and Christos Faloutsos, "The R^+-tree: A Dynamic Index for Multi-dimensional Objects", Proceedings of the Thirteenth VLDB Conference, pp 507-518, Brighton, England, September 1987.

[ULL88]
Jeff Ullman, Principles of Database and Knowledge-Base Systems, Vol. I, Computer Science Press, Rockville, Maryland, 1988.

APPENDIXES

APPENDIX A

PROGRAM LISTING

```
/*****************************************************************
                          R*- TREE
                          --------
    The R*-tree is built using the idea and algorithms as given in the
following research paper :-

Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, Bernhard
Seeger, "The R*-tree: An Efficient and Robust Access Method for
Points and Rectangles", Proc. ACM-SIGMOD International Conference
on the Management of Data, pp 322-331, 1990.

The index records are present in the leaf nodes containing pointers
to data objects. Each spatial object consists of a bounding box which
describes the extent of the object in that dimension. Bounding box is
nothing but a rectangle which has coordinates representing the lower
left x and y values and the lengths along each of these axes. Child
pointers are used for intermediate nodes in the tree.

The limits on the maximum values of coordinates is specified beforehand
so that the random generation of coordinates stays within bounds.
Insertion, accessing and deletion operations are performed. They are
verified by breadth and depth first traversals of the tree. Statistics
is also taken as regards covering parameters, number of rectangles,
number of nodes and number of leaves in the tree. Covering parameter is
the ratio of the sum of the areas of rectangles in the tree to the area
of the maximum bounding region. The user is asked to specify a covering
parameter, which should be at least 0.01, along with a name to identify
a R*-tree.
*****************************************************************/
#include <iostream.h>
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>

/* all constant definitions are declared here */
const int    MAXRECT   = 5;
const int    MINRECT   = MAXRECT / 2;
const int    MAX       = 15;
const int    MAXLEVEL  = 20;
const int    SIZE      = ((MAXRECT+1)*(MAXRECT)/2);
const int    YES       = 1;
const int    NO        = 0;
const double DEL       = 0.01;
const int    MAXN      = 4000;
const int    TOTR      = 6000;
const float  MULTIPLE  = 0.2;
const int    FOUR      = 4;
const int    DIST      = MAXRECT - 2*MINRECT + 2;
const double RIP       = 0.3;


int   ACCESS_COUNT = 0;
long  seed = 1;
float MINX = 0.0;
float MINY = 0.0;
float MAXX = 2700.0;
float MAXY = 2100.0;

/* a structure to hold the coordinates of a rectangle in an R node    */
class RECT {
  public:
```

```
        float       xlow;                /* lower coordinate of x          */
        float       ylow;                /* lower coordinate of y          */
        float       length_x;            /* length along  x axis           */
        float       length_y;            /* length along  y axis           */

        float  Area () {
        /* check for any errors in coordinates of x and y  */
        if (length_x < 0.0 || length_y < 0.0) {
           printf ("\n Length of the rectangle is negative\n");
           exit (0);
           }
        return (length_x * length_y);
        }

        float Margin () {
        if (length_x < 0.0 || length_y < 0.0) {
           printf ("\n Length of the rectangle is negative\n");
           exit (0);
           }
        return ((length_x * 2.0) + (length_y * 2.0));
        }

        int is_intersect (RECT r2) {
        int flag = 0;
        if (((xlow + length_x) > r2.xlow) &&
            ((ylow + length_y) > r2.ylow))
           if (((r2.xlow + r2.length_x) > xlow) &&
               ((r2.ylow + r2.length_y) > ylow))
              flag = 1;
        return flag;
        }

        int   coord_check () {
        int   flag = YES;
        if (length_x < MULTIPLE || length_y < MULTIPLE) {
           printf ("\nLength Parameters should be at least 0.2");
           printf ("\n %f and %f ", length_x, length_y);
           flag = NO;
           }
        if (xlow < MINX || ylow < MINY || xlow + length_x > MAXX ||
            ylow + length_y > MAXY) {
           printf ("\nRectangle coordinates beyond limit");
          printf ("\n %f %f %f %f", xlow, ylow, xlow+length_x,
                   ylow+length_y);
           flag = NO;
           }
        return flag;
        }
};

/* structure containing information for each node in R*-tree          */
class RSNODE {
  public:
        int         nofrect;             /* number of rectangle in node  */
        RECT        rect [MAXRECT];      /* set of rectangle dimensions  */
        RSNODE      *father;             /* pointer to parent node       */
        RSNODE      *child_ptr [MAXRECT]; /* array of child pointers     */
};

/* a structure head which can hold multi R*-trees is used here. Each
   R*-tree it holds has various information associated with it. */
class HEAD {
```

```
  public:
       char        rstreename [MAX];   /* name to identify an R*-tree   */
       int         total_nodes;        /* total number of nodes in tree */
       int         total_rect;         /* total number of rectangles    */
       int         htree;              /* height of the R*-tree         */
       int         M;                  /* Maximum no of rects in a node */
       int         m;                  /* Minimum no of rects in a node */
       int         level [MAXLEVEL];   /* level no for overflow check   */
       float       cover_par_actual;   /* actual covering parameter     */
       float       cover_par_assign;   /* assigned covering parameter   */
       float       area_bound;         /* total area of bounded region  */
       float       area_all_rects;     /* total area of all rectangles  */
       RSNODE      *root;              /* pointer to root of an R*-tree */
       HEAD        *next;              /* pointer to the next R*-tree   */
};

/* a temporary structure used for split routine  */
class SET {
  public:
       RECT    rect_arr [MAXRECT + 1];
       RSNODE  *child_arr [MAXRECT + 1];
};

/* Function prototypes */
int     INSERT (HEAD*, RECT);
int     SEARCH (HEAD*, RECT, RSNODE**, int*);
int     create_head (char []);
int     total_leaves (HEAD*);
int       insertion_into_rstree (HEAD*, RSNODE*, RECT, RSNODE**, RECT*,
int*);
int     Deletion (HEAD*, RECT);
int     select_entry_in_node (RECT, RSNODE*, int*);
void    interactive_program_execution (char *[]);
void    inter_access ();
void    inter_breadth ();
void    inter_create (HEAD**);
void    inter_delete ();
void    inter_depth ();
void    inter_get_id (HEAD**);
void    inter_insert (char *[]);
void    inter_kill ();
void    inter_statis ();
void    creation (HEAD**);
void    create_rect (RECT*);
void    DELETE (HEAD*, RECT);
void    KILL (HEAD*);
void    enq_rect (HEAD*, RSNODE*, int*, RECT []);
void    adjust_rect (RECT [], RSNODE* [], int, int);
void    adjust_one_rect (RSNODE*, int);
void    put_rect_in_leaf (RECT, RSNODE*, RSNODE*, int);
void    destroy_tree (RSNODE*);
void    calculate_total_leaves (RSNODE*, int*);
void    enqueue (HEAD*, RSNODE*, RECT [], int*);
void    REINSERT (HEAD*, RECT [], int, int);
void    SPLIT_Q (RECT, RSNODE*, RSNODE**, RECT*, RSNODE**);
void    statistics (HEAD*);
void    end_program ();
void    put_node (RSNODE*, RECT, RSNODE*, int);
void    bound_rect (RECT, RECT, RECT*);
void    bft_rstree (RSNODE*);
void    breadth_first_traversal (HEAD*);
void    dft_rstree (RSNODE*);
```

```
void    depth_first_traversal (HEAD*);
RECT    bounded_box (RSNODE*);
HEAD    *get_rstree (char []);
RSNODE  *create_root (RECT, RSNODE*, RSNODE*);
float   Random_no_gen ();
float   overlap (RSNODE*, int);
void    intersect (RECT, RECT, RECT*);
float   middle (float, float, float);
int     is_child_leaves (RSNODE*);
int     determine_min_overlap_cost (RECT, RSNODE*, int*);
float   Compute_S (RECT [], float [], float [], float []);
void    bub_sort (SET [], int);
int     Insert_Data (HEAD*, RECT);
int     DInsertion_rstree (HEAD*, RSNODE*, RECT, RSNODE**, RECT*, int*);
void    ReInsert (HEAD*, RSNODE**, RECT, RSNODE*);
void    sort_bubble (float []);
int     first_call_overflow_trmt (HEAD*, RSNODE*);


HEAD   *treelist_hd = NULL;

/*****************************************************************
 Main program that asks the user to type 1 for generation of
 coordinates through interactive program execution.
 Otherwise 2 to exit the program
*****************************************************************/
void main (int argc, char *argv[])
{
int   i;
int   selection;
char number[MAX];
/* prime the random numbers by calling 300 times */
for (i = 0; i < 300; i++)
     Random_no_gen ();
if (argc < 3) {
    printf ("Error in command line argument: a.out vlsidata* *out\n");
    exit (0);
    }
do {
    system ("tput clear");
    system ("tput cup 2 0");
    printf ("\n\n");
    printf ("\t\t\t1.  RUN INTERACTIVE PROGRAM\n");
    printf ("\t\t\t2.  EXIT \n\n\n");
    printf ("\t\t\t    Make your selection: \t");
    gets (number);
    selection = atoi (number);
    switch (selection) {
           case  1   :  interactive_program_execution (argv);
                        break;

           case  2   :  end_program ();
                        system ("tput clear");
           }
    if (selection < 1 || selection > 2) {
       printf ("\n\n Invalid selection !!");
       system ("sleep 1");
       selection = 1;
       }
    } while (selection == 1);
}   /* end of main program */
```

```
/*****************************************************************
 random number generation taking seed as global value of 1.0
 Obtained from:
 Stephen K. Park, and Keith W. Miller, "Random Number Generators:
 Good Ones are Hard to Find", Communications of the ACM, Vol. 31, 2,
 (October 1988), pp 1192-1201.
*****************************************************************/
float Random_no_gen ()
{
float  a, m, q, r, lo, hi, test;
a = 16807.0;  m = 2147483647.0;  q = 127773.0;   r = 2836.0;
hi = (int)(seed/q);
lo = seed - q * hi;
test = a * lo - r * hi;
if (test > 0.0)
   seed = test;
else
   seed = test + m;
return seed/m;
} /* end of Random_no_gen function */

/*****************************************************************
 Routine governs the operations performed on R*-tree. A user friendly
 menu is displayed before any operation is selected
*****************************************************************/
void interactive_program_execution (char *argv[])
{
int   choice;
char  str[MAX];
HEAD  *treeid;
do {
   system ("tput clear");
   system ("tput cup 2 0");
   printf ("\t\t\t          R*-TREE\n");
   printf ("\t\t\t          -------\n\n");
   printf ("\t\t\t1. CREATE HEADER\n");
   printf ("\t\t\t2. INSERT RECTANGLE\n");
   printf ("\t\t\t3. DELETE RECTANGLE\n");
   printf ("\t\t\t4. ACCESS LEAF RECORD\n");
   printf ("\t\t\t5. BREADTH FIRST TRAVERSAL\n");
   printf ("\t\t\t6. DEPTH FIRST TRAVERSAL\n");
   printf ("\t\t\t7. STATISTICS\n");
   printf ("\t\t\t8. KILL R*-TREE\n\n\n");
   printf ("\t\t\t   Make your choice : \t");
   gets (str);
   choice = atoi (str);
   switch (choice) {
          case  1  :  inter_create (&treeid);
                      break;

          case  2  :  inter_insert (argv);
                      break;

          case  3  :  inter_delete ();
                      break;

          case  4  :  inter_access ();
                      break;

          case  5  :  inter_breadth ();
                      break;
```

```
            case  6  :  inter_depth ();
                        break;

            case  7  :  inter_statis ();
                        break;

            case  8  :  inter_kill ();
                        break;
            }
   if (choice < 1 || choice > 8) {
      printf ("\n\n Invalid Choice !!");
      system ("sleep 1");
      choice = 1;
      }
   } while (choice >= 1 && choice < 8);
} /* end of interactive_program_execution function */

/*****************************************************************
 This routine is called by interactive program execution
 to create a header for the R*-tree
*****************************************************************/
void inter_create (HEAD **treeid)
{
char  name [MAX];
system ("tput cup 15 10");
printf ("\nCREATING TREE HEADER");
do {
   system ("sleep 1");
   system ("tput clear");
   do {
      system ("tput clear");
      printf ("\nEnter the ID of the R*tree (1 - 14 chars) : \t");
      gets (name);
      if (strcmp (name, "") == 0) {
         printf ("\nEnter a valid tree id\n\n\n");
         system ("sleep 1");
         }
      } while (strcmp (name, "") == 0);
   } while (!create_head (name));
if (((*treeid) = get_rstree (name)) == NULL) {
   printf ("\nCant locate tree with id = %s\n", name);
   exit (0);
   }
} /* end of inter_create function */

/*****************************************************************
 This routine is called by interactive program execution
 to insert rectangles into the R*-tree using vlsi data set
*****************************************************************/
void inter_insert (char *argv[])
{
char  str[MAX];
HEAD  *treeid;
RECT  rect;
FILE  *fp, *fp2;
int   i, count, cnt, j, duplicate;
char  st [80], temp [20], ch;
float x1[TOTR], x2[TOTR], y1[TOTR], y2[TOTR];
if ((fp = fopen (argv[1], "r")) == NULL) {
   printf ("Error : opening file %s\n", argv[1]);
   exit (0);
   }
```

```
if ((fp2 = fopen (argv[2], "w")) == NULL) {
   printf ("Error : opening file for write: %s\n", argv[2]);
   exit (0);
   }
if (treelist_hd == NULL) {
   printf ("R*-tree not created \n");
   fflush (stdout);
   return;
   }
system ("tput cup 15 10");
printf ("\nINSERTING RECTANGLES FROM %s", argv[1]);
system ("sleep 1");
inter_get_id (&treeid);
str[0] = 'y';
printf
("\nDo you wish to insert all rectangles given in %s (y/n):\t", argv[1]);
gets (str);
if (str[0] == 'y' || str[0] == 'Y') {
   count = 0;
   while (fgets (st, 80, fp) != NULL) {
     if (st[0] == 'r' && st[1] == 'e' && st[2] == 'c' &&  st[3] == 't'
         && st[4] == ' ') {
        cnt = 0;
        for (i = 5; i < strlen (st); i++)
            temp[cnt++] = st[i];
        temp[cnt] = '\0';
        fprintf (fp2, "%s", temp);
        count++;
        }
     }
   fclose (fp);
   fclose (fp2);
   if ((fp2 = fopen (argv[2], "r")) == NULL) {
      printf ("Error : opening file for read: %s\n", argv[2]);
      exit (0);
      }
   for (i = 0; i < count; i++) {
       fscanf (fp2, "%f %f %f %f%c", &x1[i], &y1[i], &x2[i], &y2[i], &ch);
       rect.xlow = x1[i];
       rect.ylow = y1[i];
       rect.length_x = x2[i] - x1[i];
       rect.length_y = y2[i] - y1[i];
       duplicate = NO;
       for (j = i-1, i > 0; j >= 0; j--) {
           if (x1[j] == x1[i] && y1[j] == y1[i] &&
               x2[j] == x2[i] && y2[j] == y2[i]) {
              duplicate = YES;
              break;
              }
           }
       if (rect.length_x <= 0.0 || rect.length_y <= 0.0)
          printf ("\nError in input data retrieval  !!!!\n");
       if (!duplicate && (INSERT (treeid, rect) == YES))
          treeid->area_all_rects += rect.Area ();
       }
 printf ("\nInsertion of rectangles is complete");
 printf ("\nPress Enter to continue...");      getchar ();
    fclose (fp2);
    }
 printf ("\n\n\n\n");
 } /* end of inter_insert function */
```

```
/*********************************************************************
 This routine is called by interactive program execution
 to free the pointers given the id of R*-tree and delete the header.
*********************************************************************/
void inter_kill ()
{
HEAD   *treeid;
if (treelist_hd == NULL) {
   printf ("R*-tree not created !!!");
   fflush (stdout);
   system ("sleep 1");
   return;
   }
system ("tput cup 15 10");
printf ("\nKILLING R*-TREE");
system ("sleep 1");
inter_get_id (&treeid);
KILL (treeid);
printf ("\nR*-tree killed");
fflush (stdout);
system ("sleep 1");
} /* end of inter_kill function */

/*********************************************************************
 This routine is called by interactive program execution
 to display the statistics for the given header id of R*-tree
*********************************************************************/
void inter_statis ()
{
HEAD *treeid;
if (treelist_hd == NULL) {
   printf ("R*-tree not created!!!");
   fflush (stdout);
   system ("sleep 1");
   return;
   }
system ("tput cup 15 10");
printf ("\nSTATISTICS OF R*-TREE");
system ("sleep 1");
inter_get_id (&treeid);
statistics (treeid);
printf ("\n\nPress Enter to continue...");
getchar ();
} /* end of inter_statis function */

/*********************************************************************
 This routine is called by interactive program execution
 to obtain a header name from user and check for duplicates
*********************************************************************/
void inter_get_id (HEAD **treeid)
{
char  name [MAX];
int   notok = NO;
do {
   system ("tput clear");
   printf ("\nEnter the ID of the R*-tree (1-14chars):\t");
   gets (name);
   if (((*treeid) = get_rstree (name)) == NULL) {
      printf ("\nR*-tree with id = %s does not exist\n", name);
      system ("sleep 1");
      notok = YES;
      }
```

```
   else
      notok = NO;
   } while (notok);
} /* end of inter_get_id function */

/*************************************************************************
 This routine is called by interactive program execution
 to traverse the R*-tree using depth-first search
*************************************************************************/
void inter_depth ()
{
HEAD   *treeid;
if (treelist_hd == NULL) {
   printf ("R*-tree not created \n");
   fflush (stdout);
   return;
   }
system ("tput cup 15 10");
printf ("\nDEPTH FIRST TRAVERSAL OF R*-TREE");
system ("sleep 1");
inter_get_id (&treeid);
printf ("\n");
depth_first_traversal (treeid);
printf ("\n\nPress Enter to continue...");
getchar ();
} /* end of inter_depth function */

/*************************************************************************
 This routine is called by interactive program execution
 to delete rectangles that are provided by the user. The existence
 of the rectangle is checked first before deletion.
*************************************************************************/
void inter_delete ()
{
char  str[MAX];
HEAD  *treeid;
RECT  rect;
int   notok = NO;
if (treelist_hd == NULL) {
   printf ("R*tree not created!!!!");
   fflush (stdout);
   return;
   }
system ("tput cup 15 10");
printf ("\nDELETING RECTANGLES FROM R*-TREE");
system ("sleep 1");
inter_get_id (&treeid);
do {
   str[0] = 'y';
   notok = NO;
   printf
   ("\nEnter Rectangle Coordinates(xlow, ylow, length_x, length_y):
");
   scanf ("%f %f %f %f", &rect.xlow, &rect.ylow, &rect.length_x,
                         &rect.length_y);
   getchar ();
   if (!rect.coord_check ()) {
      notok = YES;
      }
   else {
      DELETE (treeid, rect);
      printf ("\nDo you wish to delete another Rectangle (y/n):\t");
```

```
      gets (str);
      }
   } while (str[0] == 'y' || str[0] == 'Y' || notok);
} /* end of inter_delete function */

/*********************************************************************
 This routine is called by interactive program execution
 to traverse the R*-tree using breadth-first search
*********************************************************************/
void   inter_breadth ()
{
HEAD   *treeid;
if (treelist_hd == NULL) {
   printf ("R*tree is not created!!!");
   fflush (stdout);
   return;
   }
system ("tput cup 15 10");
printf ("\nBREADTH FIRST TRAVERSAL OF R*-TREE");
system ("sleep 1");
inter_get_id (&treeid);

breadth_first_traversal (treeid);
printf ("\n\nPress Enter to Continue ...");
getchar ();
} /* end of inter_breadth function */

/*********************************************************************
 This routine is called by interactive program execution
 to access a given rectangle in R*-tree. Besides searching for an
 exact match, all leaf records which overlap the given search area
 are produced.
*********************************************************************/
void   inter_access ()
{
char   str [MAX];
HEAD   *treeid;
RSNODE  *node;
RECT   rect;
int    notok = NO;
int    pos;
if (treelist_hd == NULL) {
   printf ("R_tree not created!!!");
   fflush (stdout);
   return;
   }
system ("tput cup 15 10");
printf ("\nACCESSING RECTANGLES IN R*-TREE");
system ("sleep 1");
inter_get_id (&treeid);
do {
   str[0] = 'y';
   notok = NO;
   printf ("\nEnter the rectangle coordinates(xlow ylow length_x
length_y):\t");
   scanf ("%f %f %f %f", &rect.xlow, &rect.ylow, &rect.length_x,
          &rect.length_y);
   getchar ();
   if (!rect.coord_check ()) {
      notok = YES;
      }
   else {
```

```
      if (SEARCH (treeid, rect, &node, &pos) == YES) {
         printf ("\nExact match leaf rectangle found ");
         printf ("\nNo of nodes visited = %d", ACCESS_COUNT);
         }
      else
         printf ("\nexact match rectang not found   !!!!\n");
      printf ("\nDo you wish to access another rectangle (y/n):\t");
      gets (str);
      }
   } while (str[0] == 'y' || str[0] == 'Y' || notok);
} /* end of inter_access function */

/*********************************************************************
 Rectangles are generated at random using a covering parameter. All
 operations to be done on the R*-tree are verified.
*********************************************************************/
void random_program_execution ()
{
int  cnt = 0;
int  ndx = 0;
int  max;
RECT rect;
HEAD *Head;
RSNODE *node;
RECT  r[TOTR];
int   i = 0;
max = 5;
system ("tput cup 10 10");
printf ("\nEnter max. bounding region coordinates\n");
printf ("in floating point (MINX, MINY, MAXX, MAXY): \t");
scanf ("%f %f %f %f", &MINX, &MINY, &MAXX, &MAXY);
getchar ();
/* create an r tree with the user asked for name identity */
creation (&Head);
system ("tput clear");
/* insert rectangles into the r tree */
while (Head->cover_par_actual < Head->cover_par_assign) {
      create_rect (&rect);
      Head->area_all_rects += rect.Area ();
      if ((Head->area_all_rects/Head->area_bound) >
          (Head->cover_par_assign + DEL)) {
         do {
            Head->area_all_rects -= rect.Area();
            create_rect (&rect);
            Head->area_all_rects += rect.Area ();
            } while (Head->area_all_rects / Head->area_bound >
                     (Head->cover_par_assign + DEL));
         }
     printf ("%.1f, %.1f, %.1f, %.1f\n", rect.xlow, rect.ylow,
            rect.length_x, rect.length_y);
     r[cnt].xlow = rect.xlow;
     r[cnt].ylow = rect.ylow;
     r[cnt].length_x = rect.length_x;
     r[cnt].length_y = rect.length_y;
     if (INSERT (Head, rect) == YES) {
        printf ("Rectangle is successfully inserted\n");
        Head->cover_par_actual =
                 Head->area_all_rects/Head->area_bound;
        printf ("Covering parameter = %.2f\n\n",
                Head->cover_par_actual);
        cnt++;
        }
```

```
     }
if (cnt < max)
   max = cnt;
printf ("\nBreadth First Traversal of --> %s\n", Head->rstreename);
printf ("==========================\n\n");
breadth_first_traversal (Head);
printf ("\nDepth First Traversal of --> %s\n", Head->rstreename);
printf ("==========================\n\n");
depth_first_traversal (Head);
printf ("\nAccessing Certain Rectangles in --> %s\n", Head->rstreename);
printf ("==========================\n\n");
for (i = 0; i < max; i++) {
    printf ("\n%.1f, %.1f, %.1f, %.1f",
            r[i].xlow, r[i].ylow, r[i].length_x,
            r[i].length_y);
    if (SEARCH (Head, r[i], &node, &ndx) == YES) {
       printf ("\nExact match Leaf Rectangle is    FOUND\n");
       printf ("No of nodes visited = %d\n", ACCESS_COUNT);
       }
    else
       printf ("\nRectangle is    NOT FOUND\n");
    }
printf ("\nBefore DELETION of --> %s", Head->rstreename);
statistics (Head);
printf ("\n\n\nDeleting All Rectangles in --> %s\n", Head->rstreename);
printf ("==========================\n\n");
for (i = 0; i < cnt; i++) {
    printf ("\n%.1f, %.1f, %.1f, %.1f",
            r[i].xlow, r[i].ylow, r[i].length_x,
            r[i].length_y);
    DELETE (Head, r[i]);
    }
printf ("\nAfter Deletion Breadth First Traversal ");
printf ("of --> %s\n", Head->rstreename);
printf ("==========================\n\n");
breadth_first_traversal (Head);
KILL (Head);
} /* end of random_program_execution function */

/*********************************************************************
 An R*-tree is created given the id of the tree.
*********************************************************************/
void creation (HEAD **rstreename)
{
char  str[MAX];
float  cov;
system ("tput cup 15 10");
printf ("\nCreating Tree Header");
do {
   system ("sleep 1");
   system ("tput cup 15 10");
   printf ("\n\n");
   do {
      system ("tput cup 15 10");
      printf ("\nEnter the ID of the R*tree (1 - 14 chars) : \t");
      gets (str);
      if (strcmp (str, "") == 0) {
         printf ("\nEnter a valid tree id");
         system ("sleep 1");
         system ("tput cup 15 10");
         printf ("\n\n\n");
         }
```

```
      } while (strcmp (str, "") == 0);
   } while (!create_head (str));
if (((*rstreename) = get_rstree (str)) == NULL) {
   printf ("\nCant locate tree with id = %s\n", str);
   exit (0);
   }
system ("tput cup 15 10");
printf ("\n\n\n\n");
do {
   system ("tput cup 15 10");
   printf ("\nEnter the covering parameter :\t");
   scanf ("%f", &cov);
   getchar ();
   if (cov < DEL) {
      printf ("\nCovering parameter should be greater than %.2f", DEL);
      system ("sleep 1");
      system ("tput cup 15 10");
      printf ("\n\n\n");
      }
   } while (cov < DEL);
(*rstreename)->cover_par_assign = cov;
} /* end of creation function */

/*********************************************************************
 creation of rectangles for an rsnode
*********************************************************************/
void create_rect (RECT *rect)
{
int   tmp1;
float  tmp2;
float  tmp3;
tmp3 = MAXX / 100.0;
tmp1 = (int)((Random_no_gen () * (MAXX - MINX) + tmp3) * MAXY);
tmp1 += tmp1 % 2;
tmp2 = (float)(tmp1) / MAXY;
rect->xlow = fmod (tmp2, (MAXX - MINX - MULTIPLE));
tmp1 = (int)((Random_no_gen () * (MAXY - MINY) + tmp3) * MAXY);
tmp1 += tmp1 % 2;
tmp2 = (float)(tmp1) / MAXY;
rect->ylow = fmod (tmp2, (MAXY - MINY - MULTIPLE));
do {
   tmp1 = (int)((Random_no_gen () * (MAXX - rect->xlow) + tmp3) * MAXY);
   tmp1 += tmp1 % 2;
   tmp2 = (float)(tmp1) / MAXY;
   rect->length_x = fmod (tmp2, (MAXX - rect->xlow));
   } while (rect->length_x < MULTIPLE ||
            (rect->xlow + rect->length_x) > MAXX);
do {
   tmp1 = (int)((Random_no_gen () * (MAXY - rect->ylow) + tmp3) * MAXY);
   tmp1 += tmp1 % 2;
   tmp2 = (float)(tmp1) / MAXY;
   rect->length_y = fmod (tmp2, (MAXY - rect->ylow));
   } while (rect->length_y < MULTIPLE ||
            (rect->ylow + rect->length_y) > MAXY);
} /* end of create_rect function */

/*********************************************************************
 Function to insert rectangles into the R*-tree
*********************************************************************/
int   INSERT (HEAD *Head, RECT rect)
{
int   promoted;
```

```
RECT  p_rect;
RSNODE *p_node;
int   done = NO;
int  k;
/* error check */
if (Head == NULL) {
   printf ("\n There is no R*-tree\n");
   return NO;
   }
if (Head->root == NULL) {  /* check for an R*-tree */
   if ((Head->root = (RSNODE *)malloc (sizeof (RSNODE))) == NULL) {
      printf ("\nError creating rstree node\n");
      exit (0);
      }
   memset (Head->root, 0, (MAXRECT * sizeof (RECT) + sizeof (int)));
   Head->root->father = NULL;
   for (k = 0; k < MAXRECT; k++)
       Head->root->child_ptr[k] = NULL;
   /* assigning values to the particular node in tree */
   Head->root->rect[0].xlow = rect.xlow;
   Head->root->rect[0].ylow = rect.ylow;
   Head->root->rect[0].length_x = rect.length_x;
   Head->root->rect[0].length_y = rect.length_y;
   Head->root->nofrect = 1;
   Head->total_nodes++;
   Head->total_rect++;
   Head->htree++;
   done = YES;
   }
else {
   promoted = insertion_into_rstree (Head, Head->root,
                  rect, &p_node, &p_rect, &done);
   if (promoted) {  /* promotion caused root to split */
      Head->root = create_root (p_rect, Head->root, p_node);
      Head->total_nodes++;
      Head->htree++;
      }
   }
return done;
} /* end of INSERT function */

/*********************************************************************
 Function to insert rectangles into the tree and to inform if
 split is performed or has been appropriately inserted.
*********************************************************************/
int    insertion_into_rstree (HEAD *Head, RSNODE *node, RECT rect,
                   RSNODE **p_rchild, RECT *p_rect, int *inserted)
{
RSNODE  *p_b_node;
int    flag, promoted, ndx;
RECT   p_b_rect, brect;
/* check if pointer is pointing to actual spatial object */
/* leaf check. If N is a leaf then return N.             */
if (node == NULL) {
   p_rect->xlow = rect.xlow;
   p_rect->ylow = rect.ylow;
   p_rect->length_x = rect.length_x;
   p_rect->length_y = rect.length_y;
   (*p_rchild) = NULL;
   Head->total_rect++;
   return YES;
   }
```

```
/* select the right entry to be placed in node */
/* choose subtree which is the sub function in choose leaf */
/* check if the child pointers in N point to leaves */
if (is_child_leaves (node))
   flag = determine_min_overlap_cost (rect, node, &ndx);

/* if the child pointers in N do not point to leaves */
else
   flag = select_entry_in_node (rect, node, &ndx);
/* checking for duplication errors */
if (flag && node->child_ptr[ndx] == NULL) {
   printf ("\nRectangle with same dimensions exist");
   return NO;
   }
/* recursively call and check if promotion occurs.                */
/* descend until leaf is reached  as said in choose leaf function */
/* set N to be the child node pointed to by F.p and repeat.       */
promoted = insertion_into_rstree (Head, node->child_ptr[ndx], rect,
                        &p_b_node, &p_b_rect, inserted);
if (node->child_ptr[ndx] != NULL) {
   brect = bounded_box (node->child_ptr[ndx]);
   node->rect[ndx].xlow = brect.xlow;
   node->rect[ndx].ylow = brect.ylow;
   node->rect[ndx].length_x = brect.length_x;
   node->rect[ndx].length_y = brect.length_y;
   }
if (!promoted)
   return NO;
/* insert if possible in the node otherwise split */
/* if L has room for another entry insert E. */
if (node->nofrect < MAXRECT) {
   put_rect_in_leaf (p_b_rect, p_b_node, node, ndx);
   *inserted = YES;
   return NO;
   }
/* otherwise invoke splitnode to obtain a new node LL.  Distribute */
/* E and all the old entries of L to L and LL                      */
else {  /* split is performed when node is overfull */
   SPLIT_Q (p_b_rect, p_b_node, &node, p_rect,
          &(*p_rchild));
   Head->total_nodes++;
   *inserted = YES;
   return YES;
   }
} /* end of insertion_into_rstree function */

/*****************************************************************
 function checks if the child pointers in the node point to leaves
*****************************************************************/
int is_child_leaves (RSNODE *node)
{
int i, flag = YES;
/* error checking */
if (!node) {
   printf ("\nis child leaves: The node does not exist\n");
   exit (0);
   }
for (i = 0; i < node->nofrect; i++) {
    if (node->child_ptr[i] != NULL)
       flag = NO;
    }
return flag;
```

```
} /* end of is_child_leaves function */

/*****************************************************************
 function to find nearly minimum overlap cost.  Sort the
 rectangles in N in increasing order of their area enlargement
 needed to include the new data rectangle.  Let A be the group
 of the first p entries. From the entries in A, considering all
 entries in N, choose the entry whose rectangle needs least overlap
 enlargement. Resolve ties by choosing the entry whose rectangle
 needs least area enlargement then the entry with the rectangle of
 smallest area.
*****************************************************************/
int determine_min_overlap_cost (RECT rect, RSNODE *node, int *indx)
{
int   i, j, done = NO;
float A[MAXRECT], O[MAXRECT], temp, min_o;
RECT  brect, orect;
/* error checking */
if (!node) {
   printf ("\nThe node does not exist\n");
   exit (0);
   }
/* bubble sort the rectangles in N in increasing order of their */
/* area enlargement                                             */
for (i = 0; i < node->nofrect; i++) {
    if (rect.xlow == node->rect[i].xlow &&
        rect.ylow == node->rect[i].ylow &&
        rect.length_x == node->rect[i].length_x &&
        rect.length_y == node->rect[i].length_y)
       done = YES;
    bound_rect (rect, node->rect[i], &brect);
    A[i] = brect.Area () - node->rect[i].Area ();
    }
for (i = 0; i < node->nofrect; i++) {
    for (j = i+1; j <= node->nofrect; j++) {
        if (A[i] > A[j]) {
           temp = A[i];
           A[i] = A[j];
           A[j] = temp;
           }
        }
    }
done = NO;
/* from entries in A, considering all entries in N, choose entry */
/* whose rectangle needs least overlap enlargement               */
for (i = 0; i < node->nofrect; i++) {
    if (rect.xlow == node->rect[i].xlow &&
        rect.ylow == node->rect[i].ylow &&
        rect.length_x == node->rect[i].length_x &&
        rect.length_y == node->rect[i].length_y)
       done = YES;
    O[i] = 0;
    if (rect.is_intersect (node->rect[i])) {
       intersect (rect, node->rect[i], &orect);
       O[i] = orect.Area () - node->rect[i].Area ();
       }
    }
min_o = O[0];
*indx = 0;
for (i = 1; i < node->nofrect; i++) {
    if (O [i] < min_o) {
       min_o = O [i];
```

```
        *indx = i;
        }
/* resolve ties by choosing the entry whose rectangle needs */
/* least area enlargement and then entry with rectangle of  */
/* smallest area                                            */
    else if (O [i] == min_o) {
        if (A [i] < A [*indx]) {
           min_o = O[i];
           *indx = i;
           }
        else if (A[i] == A [*indx]) {
           if (node->rect[i].Area () < node->rect[*indx].Area ()) {
              min_o = O[i];
              *indx = i;
              }
           }
        }
    }
return done;
} /* end of determine_overlap_cost function */

/*********************************************************************
 let F be the entry in N whose rectangle F.I needs least
 enlargement to include E.I.   Resolve ties by choosing
 the entry with the rectangle of smallest area.
*********************************************************************/
int  select_entry_in_node (RECT rect, RSNODE *node, int *indx)
{
int   i, done = NO;
float d[MAXRECT], min_d;
RECT  brect;
/* error checking */
if (!node) {
   printf ("\nThe node does not exist\n");
   exit (0);
   }

for (i = 0; i < node->nofrect; i++) {
    if (rect.xlow == node->rect[i].xlow &&
        rect.ylow == node->rect[i].ylow &&
        rect.length_x == node->rect[i].length_x &&
        rect.length_y == node->rect[i].length_y)
       done = YES;
    bound_rect (rect, node->rect[i], &brect);
    d[i] = brect.Area () - node->rect[i].Area ();
    }
min_d = d[0];
*indx = 0;
for (i = 1; i < node->nofrect; i++) {
    if (d [i] < min_d) {
       min_d = d[i];
       *indx = i;
       }
/* resolve ties by choosing the entry with rectangle of smallest area */
    else if (d [i] == min_d) {
       if (node->rect[i].Area () < node->rect[*indx].Area ()) {
          min_d = d[i];
          *indx = i;
          }
       }
    }
return done;
```

```
} /* end of select_entry_in_node function */

/*********************************************************************
 calculate the bounding rectangle of the two given rectangles r1 and r2
*********************************************************************/
void   bound_rect (RECT r1, RECT r2, RECT *res)
{
float  ronex, roney, rtwox, rtwoy;
if (r1.length_x < 0.0 || r1.length_y < 0.0 ||
    r2.length_x < 0.0 || r2.length_y < 0.0) {
   printf ("\nLength parameter should be positive\n");
   printf ("%f %f %f %f\n", r1.length_x, r1.length_y, r2.length_x,
                            r2.length_y);
   return;
   }
/* temporary variables are used to store the coordinates and lengths */
ronex = r1.xlow + r1.length_x;
roney = r1.ylow + r1.length_y;
rtwox = r2.xlow + r2.length_x;
rtwoy = r2.ylow + r2.length_y;
/* the resultant bounding rects x and y will the smallest of
   the two given rectangles */
res->xlow = r1.xlow < r2.xlow ? r1.xlow : r2.xlow;
res->ylow = r1.ylow < r2.ylow ? r1.ylow : r2.ylow;
/* similarly the lengths will be the larger of the two */
res->length_x = ronex > rtwox ? (ronex - res->xlow): (rtwox - res->xlow);
res->length_y = roney > rtwoy ? (roney - res->ylow): (rtwoy - res->ylow);
} /* end of bound_rect function */

/*********************************************************************
 function to place the rectangle in appropriate leaf node given position
*********************************************************************/
void  put_rect_in_leaf (RECT rect, RSNODE *rchild, RSNODE *p, int  ndx)
{
int   i;
for (i = p->nofrect; i > ndx; i--) {
    p->rect[i].xlow = p->rect[i-1].xlow;
    p->rect[i].ylow = p->rect[i-1].ylow;
    p->rect[i].length_x = p->rect[i-1].length_x;
    p->rect[i].length_y = p->rect[i-1].length_y;
    p->child_ptr[i] = p->child_ptr[i-1];
    }
p->rect[i].xlow = rect.xlow;
p->rect[i].ylow = rect.ylow;
p->rect[i].length_x = rect.length_x;
p->rect[i].length_y = rect.length_y;
p->child_ptr[i] = rchild;
p->nofrect++;
} /* end of put_rect_in_leaf function */

/*********************************************************************
 Function to split when node where rectangle to be inserted becomes
 overfull
*********************************************************************/
void SPLIT_Q (RECT rect, RSNODE *rchild, RSNODE **p_old,
              RECT *p_rect, RSNODE **p_rchild)
{
SET  tmp[FOUR];
int   k;
RECT  brect;
RECT  tempr[MAXRECT+1];
RSNODE *tempn[MAXRECT + 1];
```

```
int    tot_rect;
float   A[FOUR][DIST], M[FOUR][DIST], O[FOUR][DIST];
int     i, j;
float   Sum_of_margins [FOUR], minS, minO;
int     min_ndx, ndx, cutoff;
/* store in a temp-rectangle array to be later inserted */
for (i = 0; i < MAXRECT; i++) {
    tempr[i].xlow = (*p_old)->rect[i].xlow;
    tempr[i].ylow = (*p_old)->rect[i].ylow;
    tempr[i].length_x = (*p_old)->rect[i].length_x;
    tempr[i].length_y = (*p_old)->rect[i].length_y;
    tempn[i] = (*p_old)->child_ptr[i];
    }
tempr[i].xlow = rect.xlow;
tempr[i].ylow = rect.ylow;
tempr[i].length_x = rect.length_x;
tempr[i].length_y = rect.length_y;
tempn[i] = rchild;
for (k = 0; k < MAXRECT; k++)
    (*p_old)->child_ptr[k] = NULL;
/* check for error in creation of node */
if (((*p_rchild) = (RSNODE *)malloc (sizeof (RSNODE))) == NULL) {
   printf ("\n Error in creation of tree node in split quadratic\n");
   exit (0);
   }
memset ((*p_rchild), 0, (MAXRECT * sizeof (RECT) + sizeof (int)));
(*p_rchild)->father = (*p_old)->father;
for (k = 0; k < MAXRECT; k++)
    (*p_rchild)->child_ptr[k] = NULL;
(*p_rchild)->nofrect = 0;

for (i = 0; i < FOUR; i++) {
    for (j = 0; j < (MAXRECT + 1); j++) {
        tmp[i].rect_arr[j].xlow = tempr[j].xlow;
        tmp[i].rect_arr[j].ylow = tempr[j].ylow;
        tmp[i].rect_arr[j].length_x = tempr[j].length_x;
        tmp[i].rect_arr[j].length_y = tempr[j].length_y;
        tmp[i].child_arr[j] = tempn[j];
        }
     bub_sort (tmp, i);
    }

for (i = 0; i < FOUR; i++)
    Sum_of_margins [i] = Compute_S (tmp[i].rect_arr, A[i], M[i], O[i]);
minS = Sum_of_margins[0];
min_ndx = 0;
for (i = 1; i < FOUR; i++) {
    if (Sum_of_margins[i] < minS) {
       minS = Sum_of_margins[i];
       min_ndx = i;
       }
    }

minO = O[min_ndx][0];
ndx = 0;
for (i = 1; i < DIST; i++) {
    if (O[min_ndx][i] < minO) {
       minO = O[min_ndx][i];
       ndx = i;
       }
    else if (O[min_ndx][i] == minO) {
       if (A[min_ndx][ndx] < A[min_ndx][i]) {
```

```
            minO = O[min_ndx][ndx];
            }
         else if (A[min_ndx][ndx] > A[min_ndx][i]) {
            minO = O[min_ndx][i];
            ndx = i;
            }
         }
      }

cutoff = (MINRECT - 1) + (ndx+1);     /* because ndx is zero based */
tot_rect = MAXRECT + 1;
(*p_old)->nofrect = 0;
for (i = 0; i < cutoff; i++) {
    put_node (*p_old, tmp[min_ndx].rect_arr[i],
                tmp[min_ndx].child_arr[i], (*p_old)->nofrect);
    }
for (i = cutoff; i < MAXRECT + 1; i++) {
    put_node (*p_rchild, tmp[min_ndx].rect_arr[i],
                tmp[min_ndx].child_arr[i], (*p_rchild)->nofrect);
    }
brect = bounded_box (*p_rchild);
p_rect->xlow = brect.xlow;
p_rect->ylow = brect.ylow;
p_rect->length_x = brect.length_x;
p_rect->length_y = brect.length_y;
} /* end of SPLIT_Q function */

/*********************************************************************
 function to compute the sum of margins required for split routine
*********************************************************************/
float   Compute_S (RECT rect [MAXRECT + 1], float A[DIST],
                   float M[DIST], float O[DIST])
{
RECT     tmp1, tmp2, tmp;
int      j, k;
float      Sum = 0;
for (k = 0; k < DIST; k++) {
    bound_rect (rect[0], rect[1], &tmp1);
    for (j = 2; j <= ((MINRECT-1)+k); j++) {
        bound_rect (tmp1, rect[j], &tmp1);
        }
    bound_rect (rect[j], rect[j+1], &tmp2);
    for (j = j+2; j < MAXRECT+1; j++)
        bound_rect (tmp2, rect[j], &tmp2);
    A[k] = tmp1.Area () + tmp2.Area ();
    M[k] = tmp1.Margin () + tmp2.Margin ();
    if (tmp1.is_intersect (tmp2)) {
       intersect (tmp1, tmp2, &tmp);
       O[k] = tmp.Area ();
       }
    else
       O[k] = 0;
    Sum = Sum + M[k];
    }
return Sum;
}  /* end of Compute_S function */

/*********************************************************************
 bubble sort the entries when overflow occurs necessary for split
*********************************************************************/
void   bub_sort (SET tmp [], int N)
{
```

```
int        i, j;
float      temp1, temp2, temp3, temp4;
RSNODE     *temp_child;
switch (N) {
  case 0 :        /* x lower  */
                  for (i = 0; i < MAXRECT; i++) {
                      for (j = i+1; j <= MAXRECT; j++) {
                          if    (tmp[0].rect_arr[i].xlow    >
tmp[0].rect_arr[j].xlow)
{
                              temp1 = tmp[0].rect_arr[i].xlow;
                              temp2 = tmp[0].rect_arr[i].ylow;
                              temp3 = tmp[0].rect_arr[i].length_x;
                              temp4 = tmp[0].rect_arr[i].length_y;
                                      tmp[0].rect_arr[i].xlow    =
tmp[0].rect_arr[j].xlow;
                                      tmp[0].rect_arr[i].ylow    =
tmp[0].rect_arr[j].ylow;
                              tmp[0].rect_arr[i].length_x =
tmp[0].rect_arr[j].length_x;
                              tmp[0].rect_arr[i].length_y =
tmp[0].rect_arr[j].length_y;
                              tmp[0].rect_arr[j].xlow = temp1;
                              tmp[0].rect_arr[j].ylow = temp2;
                              tmp[0].rect_arr[j].length_x = temp3;
                              tmp[0].rect_arr[j].length_y = temp4;

                              temp_child = tmp[0].child_arr[i];
                              tmp[0].child_arr[i] = tmp[0].child_arr[j];
                              tmp[0].child_arr[j] = temp_child;
                              }
                          }
                      }
                  break;

  case 1 :        /* x upper  */
                  for (i = 0; i < MAXRECT; i++) {
                      for (j = i+1; j <= MAXRECT; j++) {
                          if ((tmp[1].rect_arr[i].xlow +
tmp[1].rect_arr[i].length_x) >
                              (tmp[1].rect_arr[j].xlow +
tmp[1].rect_arr[j].length_x)) {
                              temp1 = tmp[1].rect_arr[i].xlow;
                              temp2 = tmp[1].rect_arr[i].ylow;
                              temp3 = tmp[1].rect_arr[i].length_x;
                              temp4 = tmp[1].rect_arr[i].length_y;
                                      tmp[1].rect_arr[i].xlow    =
tmp[1].rect_arr[j].xlow;
                                      tmp[1].rect_arr[i].ylow    =
tmp[1].rect_arr[j].ylow;
                              tmp[1].rect_arr[i].length_x =
tmp[1].rect_arr[j].length_x;
                              tmp[1].rect_arr[i].length_y =
tmp[1].rect_arr[j].length_y;
                              tmp[1].rect_arr[j].xlow = temp1;
                              tmp[1].rect_arr[j].ylow = temp2;
                              tmp[1].rect_arr[j].length_x = temp3;
                              tmp[1].rect_arr[j].length_y = temp4;
                              temp_child = tmp[1].child_arr[i];
                              tmp[1].child_arr[i] = tmp[1].child_arr[j];
                              tmp[1].child_arr[j] = temp_child;
                              }
```

```
                    }
                }
            break;

  case 2 :      /*  y lower  */
            for (i = 0; i < MAXRECT; i++) {
                for (j = i+1; j <= MAXRECT; j++) {
                    if    (tmp[2].rect_arr[i].ylow    >
tmp[2].rect_arr[j].ylow)
{
                        temp1 = tmp[2].rect_arr[i].xlow;
                        temp2 = tmp[2].rect_arr[i].ylow;
                        temp3 = tmp[2].rect_arr[i].length_x;
                        temp4 = tmp[2].rect_arr[i].length_y;
                            tmp[2].rect_arr[i].xlow    =
tmp[2].rect_arr[j].xlow;
                            tmp[2].rect_arr[i].ylow    =
tmp[2].rect_arr[j].ylow;
                        tmp[2].rect_arr[i].length_x =
tmp[2].rect_arr[j].length_x;
                        tmp[2].rect_arr[i].length_y =
tmp[2].rect_arr[j].length_y;
                        tmp[2].rect_arr[j].xlow = temp1;
                        tmp[2].rect_arr[j].ylow = temp2;
                        tmp[2].rect_arr[j].length_x = temp3;
                        tmp[2].rect_arr[j].length_y = temp4;
                        temp_child = tmp[2].child_arr[i];
                        tmp[2].child_arr[i] = tmp[2].child_arr[j];
                        tmp[2].child_arr[j] = temp_child;
                        }
                    }
                }
            break;

  case 3 :      /*  y upper  */
            for (i = 0; i < MAXRECT; i++) {
                for (j = i+1; j <= MAXRECT; j++) {
                    if ((tmp[3].rect_arr[i].ylow +
tmp[3].rect_arr[i].length_y) >
                        (tmp[3].rect_arr[j].ylow +
tmp[3].rect_arr[i].length_y)) {
                        temp1 = tmp[3].rect_arr[i].xlow;
                        temp2 = tmp[3].rect_arr[i].ylow;
                        temp3 = tmp[3].rect_arr[i].length_x;
                        temp4 = tmp[3].rect_arr[i].length_y;
                            tmp[3].rect_arr[i].xlow    =
tmp[3].rect_arr[j].xlow;
                            tmp[3].rect_arr[i].ylow    =
tmp[3].rect_arr[j].ylow;
                        tmp[3].rect_arr[i].length_x =
tmp[3].rect_arr[j].length_x;
                        tmp[3].rect_arr[i].length_y =
tmp[3].rect_arr[j].length_y;
                        tmp[3].rect_arr[j].xlow = temp1;
                        tmp[3].rect_arr[j].ylow = temp2;
                        tmp[3].rect_arr[j].length_x = temp3;
                        tmp[3].rect_arr[j].length_y = temp4;
                        temp_child = tmp[3].child_arr[i];
                        tmp[3].child_arr[i] = tmp[3].child_arr[j];
                        tmp[3].child_arr[j] = temp_child;
                        }
                    }
```

```
                    }
                break;

  default:
                break;
  } /* end of switch */
}   /* end of bub_sort function */

/*****************************************************************
 search for a node in R*-tree
*****************************************************************/
int     SEARCH (HEAD *Head, RECT acc_rect, RSNODE **node, int *index)
{
RSNODE  *stack[MAXN], *cur;
int     stack_ind = 0, ndx = 0, flag = NO;
float     t_x1, t_y1;
float     t_x2, t_y2;
if (Head == NULL) {
   printf ("R*tree does not exist\n");
   return NO;
   }
if (Head->root == NULL) {
   printf ("R*tree is empty.\n\n");
   return NO;
   }
cur = Head->root;
ACCESS_COUNT = 1;
do {
   for (ndx = 0; ndx < cur->nofrect; ndx++) {
        flag = NO;
        if (acc_rect.is_intersect (cur->rect[ndx]) == YES)
           flag = YES;
        else
           flag = NO;
        if (flag) {
           if (cur->child_ptr[ndx] != NULL)
              stack[stack_ind++] = cur->child_ptr[ndx];
           }
        }
   /* search for exact match in leaf level */
   if (cur->child_ptr[0] == NULL) {
      for (ndx = 0; ndx < cur->nofrect; ndx++) {
           t_x1 = acc_rect.xlow;
           t_y1 = acc_rect.ylow;
           t_x2 = cur->rect[ndx].xlow;
           t_y2 = cur->rect[ndx].ylow;
           flag = NO;
           if ((t_x1 == t_x2) && (t_y1 == t_y2))
              flag = YES;
           else
              flag = NO;
           t_x1 = acc_rect.xlow + acc_rect.length_x;
           t_x2 = cur->rect[ndx].xlow + cur->rect[ndx].length_x;
           if (flag && (t_x1 == t_x2))
              flag = YES;
           else
              flag = NO;
           t_y1 = acc_rect.ylow + acc_rect.length_y;
           t_y2 = cur->rect[ndx].ylow + cur->rect[ndx].length_y;
           if (flag && (t_y1 == t_y2))
              flag = YES;
           else
```

```
            flag = NO;
            *node = cur;
            *index  = ndx;
            if (flag)
               return YES;
            if (acc_rect.is_intersect (cur->rect[ndx]) == YES) {
               printf ("\nOverlapping leaf record : %f %f %f %f",
                        cur->rect[ndx].xlow, cur->rect[ndx].ylow,
                        cur->rect[ndx].length_x, cur->rect[ndx].length_y);
               }
            }
         }
      stack_ind--;
      if (stack_ind >= 0) {
         cur = stack[stack_ind];
         ACCESS_COUNT++;
         }
      } while (stack_ind >= 0);
return NO;
} /* end of SEARCH function*/

/*****************************************************************
 places the rectangle in the node given the position
*****************************************************************/
void   put_node (RSNODE *node, RECT rect, RSNODE *down, int ndx)
{
node->rect[ndx].xlow = rect.xlow;
node->rect[ndx].length_x = rect.length_x;
node->rect[ndx].ylow = rect.ylow;
node->rect[ndx].length_y = rect.length_y;
node->child_ptr[ndx] = down;
if (down != NULL)
   node->child_ptr[ndx]->father = node;
node->nofrect++;
} /* end of put_node function */

/*****************************************************************
 breadth first traversal of R*-tree
*****************************************************************/
void   bft_rstree (RSNODE *node)
{
RSNODE  *Q [MAXN];
int     i = 0;
int     counter = 1, level = 1;
int     f = 0, r = 0;
if (node != NULL)
   Q [f++] = node;
else {
   printf ("\nError in breadth first\n");
   exit (0);
   }
printf ("Level %d \n", level++);
printf ("--------\n");
while (f != r) {
      node = Q [r++];
      counter--;
      for (i = 0; i < MAXRECT; i++)
          if (node->child_ptr[i] != NULL)
             Q [f++] = node->child_ptr[i];
      if (node != NULL) {
         for (i = 0; i < node->nofrect; i++)
             printf ("%.1f, %.1f, %.1f, %.1f\n", node->rect[i].xlow,
```

```
                        node->rect[i].ylow, node->rect[i].length_x,
                        node->rect[i].length_y);
            printf ("\n");
            }
         if (counter == 0) {
            if (f != r) {
               printf ("Level %d \n", level++);
               printf ("--------\n");
               }
            counter = f - r;
            }
         }
} /* end of bft_rstree function */

/*********************************************************************
 calls bft_rstree after appropriate error checks
*********************************************************************/
void  breadth_first_traversal (HEAD *Head)
{
if (Head == NULL) {
   printf ("\nR*-tree does not exist\n");
   return;
   }
if (Head->root != NULL)
   bft_rstree (Head->root);
else {
   printf ("\nThe R*-tree is empty\n");
   return;
   }
return;
} /* end of breadth_first_traversal function */

/*********************************************************************
 adjust rectangle in node after deletion
*********************************************************************/
void   adjust_rect (RECT rect [], RSNODE *child [], int ndx, int tot_rect)
{
int    i;
if (ndx >= tot_rect) {
   printf ("\nPosition is not correct\n");
   exit (0);
   }
for (i = ndx; i < (tot_rect - 1); i++) {
    rect [i].xlow = rect [i+1].xlow;
    rect [i].ylow = rect [i+1].ylow;
    rect [i].length_x = rect [i+1].length_x;
    rect [i].length_y = rect [i+1].length_y;
    child [i] = child [i+1];
    }
} /* end of adjust_rect function */

/*********************************************************************
 adjust a single rectangle in node
*********************************************************************/
void  adjust_one_rect (RSNODE *node, int ndx)
{
int   i;
if (ndx >= node->nofrect) {
   printf ("\nPosition is not correct in node\n");
   exit (0);
   }
for (i = ndx; i < (node->nofrect - 1); i++) {
```

```
     node->rect[i].xlow = node->rect[i+1].xlow;
     node->rect[i].ylow = node->rect[i+1].ylow;
     node->rect[i].length_x = node->rect[i+1].length_x;
     node->rect[i].length_y = node->rect[i+1].length_y;
     node->child_ptr[i] = node->child_ptr[i+1];
     }
for ( ; i < MAXRECT; i++)
     node->child_ptr[i] = NULL;
node->nofrect--;
} /* end of adjust_one_rect function */

/*****************************************************************
 creation of a header given id and assigning memory for it
*****************************************************************/
int  create_head (char rstreename [])
{
HEAD *Head, *tmp;
if (treelist_hd != NULL) {
   tmp = treelist_hd;
   while (strcmp (tmp->rstreename, rstreename) != 0 && tmp->next != NULL)
         tmp = tmp->next;
   if (strcmp (tmp->rstreename, rstreename) == 0) {
      printf ("R*tree with header = %s already exists\n", rstreename);
      return NO;
      }
   }
if ((Head = (HEAD *)malloc (sizeof (HEAD))) == NULL) {
   printf ("Error creating header\n\n");
   exit (1);
   }
strcpy (Head->rstreename, rstreename); /* copy ids to head pointers */
Head->root = NULL;
Head->next = NULL;
Head->total_nodes = 0;
Head->total_rect = 0;
Head->htree = 0;
Head->cover_par_actual = 0.0;
Head->cover_par_assign = 0.0;
Head->area_bound = (MAXX - MINX) * (MAXY - MINY);
Head->area_all_rects = 0.0;
Head->M = MAXRECT;
Head->m = MINRECT;
if (treelist_hd == NULL)
   treelist_hd = Head;
else {
   tmp = treelist_hd;
   while (tmp->next != NULL)
         tmp = tmp->next;
   tmp->next = Head;
   }
printf ("R*_Tree with header = %s created\n", rstreename);
return YES;
} /* end of create_head function */

/*****************************************************************
 create a root for the R*-tree
*****************************************************************/
RSNODE  *create_root (RECT rect, RSNODE *rect1, RSNODE *rect2)
{
RSNODE  *tmp;
RECT   fir_rect;
int  k;
```

```
if ((tmp = (RSNODE *)malloc (sizeof (RSNODE))) == NULL) {
   printf ("\nError creating tree node \n");
   exit (0);
   }

memset (tmp, 0, (MAXRECT*sizeof(RECT)+sizeof(int)));
tmp->father = NULL;
tmp->child_ptr[0] = rect1;
tmp->child_ptr[1] = rect2;
for (k = 2; k < MAXRECT; k++)
    tmp->child_ptr[k] = NULL;
if (rect1 != NULL) {
   rect1->father = tmp;
   fir_rect = bounded_box (rect1);
   tmp->rect[0].xlow = fir_rect.xlow;
   tmp->rect[0].ylow = fir_rect.ylow;
   tmp->rect[0].length_x = fir_rect.length_x;
   tmp->rect[0].length_y = fir_rect.length_y;
   tmp->nofrect = 2;
   }
else {
   printf ("\nError in creating root - promotion.\n");
   exit (0);
   }
if (rect2 != NULL) {
   rect2->father = tmp;
   tmp->rect[1].xlow = rect.xlow;
   tmp->rect[1].ylow = rect.ylow;
   tmp->rect[1].length_x = rect.length_x;
   tmp->rect[1].length_y = rect.length_y;
   }
else {
   printf ("\nError in creating root - promotion.\n");
   exit (0);
   }
return (tmp);
} /* end of create_root function */

/*********************************************************************
 deletion of a rectangle in R*-tree
*********************************************************************/
int   Deletion (HEAD *Head, RECT rect)
{
int   f = 0, r = 0, ndx = 0, Flag = NO;
RSNODE *node, *prev;
RECT  Q [TOTR], brect;
if (SEARCH(Head, rect, &node, &ndx) == NO) {
   printf ("Rectangle not found. ");
   return NO;
   }
adjust_one_rect (node, ndx);
if (node->father == NULL) { /* case if the root is leaf */
   if (node->nofrect == 0) {
      Head->root = NULL;
      Head->total_nodes--;
      Head->htree--;
      free (node);
      }
   Head->total_rect--;
   return YES;
   }
if (node->nofrect < MINRECT) {
```

```
    enq_rect (Head, node, &f, Q);
    Flag = YES ;
    }
else
    brect = bounded_box (node);
prev = node;
node = node->father;
/* propagate till we reach root */
while (node != NULL) {
       ndx = 0;
       if (!Flag) {
          while (node->child_ptr[ndx] != prev)
                ndx++;
          node->rect[ndx].xlow = brect.xlow;
          node->rect[ndx].ylow = brect.ylow;
          node->rect[ndx].length_x = brect.length_x;
          node->rect[ndx].length_y = brect.length_y;
          }
       if (node->father != NULL) {  /* check if root */
          if (node->nofrect < MINRECT) {
             enq_rect (Head, node, &f, Q);
             Flag = YES;
             }
          else {
             brect = bounded_box (node);
             Flag = NO;
             }
          }
       prev = node;
       node = node->father;
       }
/* root has become underfull */
if (prev->nofrect < MINRECT) {
    Head->root = prev->child_ptr[0];
    if (Head->root != NULL)
       Head->root->father = NULL;
    Head->total_nodes--;
    Head->htree--;
    free (prev);
    }
REINSERT (Head, Q, f, r);
Head->total_rect--;
return YES;
} /* end of Deletion function */

/*********************************************************************
 function to insert all nodes because of propagation during deletion
 when nodes become underfull
*********************************************************************/
void enqueue (HEAD *Head, RSNODE *node, RECT Q [], int *f)
{
int    i = 0, bool = NO;
if (node != NULL) {
    if (node->child_ptr[0] == NULL)
       bool = YES;
    enqueue (Head, node->child_ptr[0], Q, f);
    enqueue (Head, node->child_ptr[1], Q, f);
    enqueue (Head, node->child_ptr[2], Q, f);
    enqueue (Head, node->child_ptr[3], Q, f);
    enqueue (Head, node->child_ptr[4], Q, f);
    if (bool == YES) {
       for (i = 0; i < node->nofrect; i++) {
```

```
            Q[*f].xlow = node->rect[i].xlow;
            Q[*f].ylow = node->rect[i].ylow;
            Q[*f].length_x = node->rect[i].length_x;
            Q[*f].length_y = node->rect[i].length_y;
            Head->total_rect--;
            (*f)++;
            }
        }
    free (node);
    Head->total_nodes--;
    }
} /* end of enqueue function */

/*********************************************************************
 Function to check if any errors, otherwise calls Deletion to
 delete a rectangle in R*-tree
*********************************************************************/
void   DELETE (HEAD *tree, RECT rect)
{
if (tree == NULL) {
    printf ("\nThe R*tree does not exist !!");
    return;
    }
if (tree->root == NULL) {
    printf ("The R*tree is empty !!");
    return;
    }
if (Deletion (tree, rect) == YES) {
    printf ("Rectangle deleted from R*_tree\n");
    tree->area_all_rects -= rect.Area ();
    tree->cover_par_actual = tree->area_all_rects/tree->area_bound;
    return;
    }
else {
    printf ("    Rectangle is not deleted!!");
    return;
    }
} /* end of DELETE function */

/*********************************************************************
 Depth first traversal of R*-tree
*********************************************************************/
void  dft_rstree (RSNODE *node)
{
RSNODE *stack[MAXN];
int     i = 0;
int     s_ndx = 0;
if (node != NULL)
    stack[s_ndx++] = node;
else {
    printf ("\nError in depth first\n");
    exit (0);
    }
while (s_ndx > 0) {
    node = stack[--s_ndx];
    for (i = 0; i < MAXRECT; i++) {
        if (node->child_ptr[i] != NULL)
            stack[s_ndx++] = node->child_ptr[i];
        }
    if (node != NULL) {
       for (i = 0; i < node->nofrect; i++)
            printf ("%.1f, %.1f, %.1f, %.1f\n", node->rect[i].xlow,
```

```
                    node->rect[i].ylow, node->rect[i].length_x,
                    node->rect[i].length_y);
        printf ("\n");
        }
    }
} /* end of dft_rstree function */

/*********************************************************************
 Check for errors before calling dft_rstree
*********************************************************************/
void   depth_first_traversal (HEAD *Head)
{
if (Head == NULL) {
   printf ("\nThe R*tree does not exist !!");
   return;
   }
if (Head->root != NULL)
   dft_rstree (Head->root);
else {
   printf ("\nThe R*-tree is empty\n");
   return;
   }
return;
} /* end of depth_first_traversal function */

/*********************************************************************
 Function to kill the header pointer after destroying all R*-trees
 if any.
*********************************************************************/
void   KILL (HEAD *Head)
{
HEAD   *tmp, *prev;
if (Head == NULL) {
   printf ("\nR*-tree does not exist\n");
   return;
   }
if (Head->root != NULL)
   destroy_tree (Head->root);
tmp = treelist_hd;
prev = treelist_hd;
while (tmp->next != NULL && tmp!= Head) {
      prev = tmp;
      tmp = tmp->next;
      }
prev->next = tmp->next;
if (prev == tmp)
   treelist_hd = NULL;
free (tmp);
} /* end of KILL function */

/*********************************************************************
 function to put rectangles in queue for later reinsertion
*********************************************************************/
void   enq_rect (HEAD *Head, RSNODE *node, int *f, RECT Q [])
{
int    i, ndx = 0;
/* check if node is a leaf */
if (node->child_ptr[0] == NULL) {
   for (i = 0; i < node->nofrect; i++) {
       Q [*f].xlow = node->rect[i].xlow;
       Q [*f].ylow = node->rect[i].ylow;
       Q [*f].length_x = node->rect[i].length_x;
```

```
        Q [*f].length_y = node->rect[i].length_y;
        Head->total_rect--;
        (*f)++;
        }
    while (node->father->child_ptr[ndx] != node)
          ndx++;
    node->father->child_ptr[ndx] = NULL;
    adjust_one_rect (node->father, ndx);
    free (node);
    Head->total_nodes--;
    return;
    }
/* node is not a leaf */
while (node->father->child_ptr[ndx] != node)
      ndx++;

node->father->child_ptr[ndx] = NULL;
adjust_one_rect (node->father, ndx);
enqueue (Head, node, Q, f);
return;
} /* end of enq_rect function */

/*****************************************************************
 given a node find the bounding region which covers all children of
 that node
*****************************************************************/
RECT  bounded_box (RSNODE *node)
{
int    i = 0;
RECT   tmp;
if (node == NULL) {
    printf ("\nNode does not exist\n");
    exit (0);
    }
if (node->nofrect == 1)
    return node->rect[0];
bound_rect (node->rect[0], node->rect[1], &tmp);
for (i = 2; i < node->nofrect; i++)
     bound_rect (tmp, node->rect[i], &tmp);
return tmp;
} /* end of bounded_box function */

/*****************************************************************
 function to find the total number of leaves in an R*-tree
*****************************************************************/
int  total_leaves (HEAD *Head)
{
int   tot_no_of_leaves = 0;
if (Head == NULL) {
    printf ("\nR Tree does not exist\n");
    return NO;
    }
if (Head->root != NULL)
    calculate_total_leaves (Head->root, &tot_no_of_leaves);
else {
    printf ("\nThe R*-tree is empty\n");
    return NO;
    }
return tot_no_of_leaves;
} /* end of total_leaves function */

/*****************************************************************
```

```
 function to get the particular R*-tree so that various operations can
 be performed on it
*********************************************************************/
HEAD  *get_rstree (char rstreename [])
{
HEAD  *temp;
if (treelist_hd == NULL) {
   printf ("\n Tree list is NULL\n");
   return NO;
   }
temp = treelist_hd;
while (strcmp (temp->rstreename, rstreename) != 0 && temp->next != NULL)
      temp = temp->next;
if (temp != NULL) {
   if (strcmp (temp->rstreename, rstreename) == 0)
      return temp;
   else
      return NULL;
   }
else
   return NULL;
} /* end of get_rstree function */

/*********************************************************************
 Function to reinsert rectangles back in R*-tree after propagation
 and underfull cases are checked while deletion
*********************************************************************/
void  REINSERT (HEAD *Head, RECT Q [], int f, int r)
{
while (r != f) {
      Insert_Data (Head, Q[r]);
      r++;
      }
} /* end of REINSERT function */

/*********************************************************************
 Function to print statistics of R*-tree
*********************************************************************/
void statistics (HEAD *treeid)
{
int  LEAF = 0;
printf ("\nSTATISTICS OF THE R*-TREE\n");
printf ("\n-----------------------\n");
printf ("\nMaximum number of rectangles in a node (M)  =  %d\n", MAXRECT);
printf ("\nMinimum number of rectangles in a node (m)  =  %d\n", MINRECT);
printf ("\nCovering Parameter (Assigned)               =  %.2f\n",
         treeid->cover_par_assign);
printf ("\nCovering Parameter (Actual)                 =  %.2f\n",
         treeid->cover_par_actual);
printf ("\nHeight of the R*-tree                       =  %d\n",
         treeid->htree);
printf ("\nNo of rectangles contained in R*-tree       =  %d\n",
         treeid->total_rect);
printf ("\nNo of nodes contained in R*-tree            =  %d\n",
         treeid->total_nodes);
if ((LEAF = total_leaves (treeid)) != NO)
printf ("\nNo of leaf nodes contained in R*-tree       =  %d\n",
         LEAF);
} /* end of statistics function */

/*********************************************************************
 function to destroy the tree and free-up the nodes in postorder fashion
```

```
*****************************************************************/
void destroy_tree (RSNODE *node)
{
if (node != NULL) {
   destroy_tree (node->child_ptr[0]);
   destroy_tree (node->child_ptr[1]);
   destroy_tree (node->child_ptr[2]);
   destroy_tree (node->child_ptr[3]);
   destroy_tree (node->child_ptr[4]);
   free (node);
   }
} /* end of destroy_tree function */

/*****************************************************************
 function to calculate total leaves in R*-tree
*****************************************************************/
void calculate_total_leaves (RSNODE *node, int *leaf)
{
if (node != NULL) {
   if (node->child_ptr[0] == NULL)
      (*leaf)++;
   calculate_total_leaves (node->child_ptr[0], leaf);
   calculate_total_leaves (node->child_ptr[1], leaf);
   calculate_total_leaves (node->child_ptr[2], leaf);
   calculate_total_leaves (node->child_ptr[3], leaf);
   calculate_total_leaves (node->child_ptr[4], leaf);
   }
} /* end of calculate_total_leaves function */

/*****************************************************************
 function to terminate the program and make sure all trees are
 killed as well as the header
*****************************************************************/
void end_program ()
{
HEAD  *Head;
if (treelist_hd != NULL) {
   while (treelist_hd->next != NULL) {
         Head = treelist_hd->next;
         treelist_hd->next = Head->next;
         KILL (Head);
         }
   KILL (treelist_hd);
   }
} /* end of end_program function */

/*****************************************************************
 function returns total area of overlapping entries in a node
*****************************************************************/
float overlap (RSNODE* node, int ndx)
{
int   i;
float total_overlap = 0.0;
RECT  tmp;
if (ndx < 0 || ndx >= node->nofrect) {
   printf ("Overlap: Position is not correct in node\n");
   exit (0);
   }
for (i = 0; i < node->nofrect; i++) {
    if (i != ndx) {
       if (node->rect[ndx].is_intersect (node->rect[i])) {
          intersect (node->rect[ndx], node->rect[i], &tmp);
```

```
            total_overlap += tmp.Area ();
            }
        }
    }
return total_overlap;
} /* end of overlap function */

/*****************************************************************
 return intersecting rectangle r3 if r1 and r2 intersect
*****************************************************************/
void intersect (RECT r1, RECT r2, RECT *r3)
{
float  xtop, ytop;
r3->xlow = middle (r1.xlow, r2.xlow, r1.xlow+r1.length_x);
r3->ylow = middle (r1.ylow, r2.ylow, r1.ylow+r1.length_y);
xtop     = middle (r1.xlow, r2.xlow+r2.length_x, r1.xlow+r1.length_x);
r3->length_x = xtop - r3->xlow;
ytop     = middle (r1.ylow, r2.ylow+r2.length_y, r1.ylow+r1.length_y);
r3->length_y = ytop - r3->ylow;
}

/*****************************************************************
 find the value that lies between the other two values.
*****************************************************************/
float middle (float a, float b, float c)
{
if ((b <= a && b >= c) || (b <= c && b >= a))
   return b;
if ((c <= a && c >= b) || (c >= a && c <= b))
   return c;
if ((a <= b && a >= c) || (a >= b && a <= c))
   return a;
}

/*****************************************************************
 reinsert rectangles back in  R*-tree after deletion
*****************************************************************/
int    Insert_Data (HEAD *Head, RECT rect)
{
int    promoted;
RECT   p_rect;
RSNODE *p_node;
int    done = NO;
int   k;
int   lcount;
/* error check */
if (Head == NULL) {
   printf ("\n There is no R*-tree\n");
   return NO;
   }
if (Head->root == NULL) {  /* check for an R*-tree */
   if ((Head->root = (RSNODE *)malloc (sizeof (RSNODE))) == NULL) {
      printf ("\nError creating rstree node\n");
      exit (0);
      }
   memset (Head->root, 0, (MAXRECT * sizeof (RECT) + sizeof (int)));
   Head->root->father = NULL;
   for (k = 0; k < MAXRECT; k++)
       Head->root->child_ptr[k] = NULL;
   /* assigning values to the particular node in tree */
   Head->root->rect[0].xlow = rect.xlow;
   Head->root->rect[0].ylow = rect.ylow;
```

```
    Head->root->rect[0].length_x = rect.length_x;
    Head->root->rect[0].length_y = rect.length_y;
    Head->root->nofrect = 1;
    Head->total_nodes++;
    Head->total_rect++;
    Head->htree++;
    done = YES;
    }
else {
    /* initialize all levels for checking as first call in overflow */
    /* treatment */
    for (lcount = 0; lcount < MAXLEVEL; lcount++)
        Head->level[lcount] = YES;
    promoted = DInsertion_rstree (Head, Head->root,
                    rect, &p_node, &p_rect, &done);
    if (promoted) {  /* promotion caused root to split */
       Head->root = create_root (p_rect, Head->root, p_node);
       Head->total_nodes++;
       Head->htree++;
       }
    }
return done;
} /* end of Insert_Data function */

/*****************************************************************
 routine to reinsert rectangles and determine if split is necessary.
*****************************************************************/
int     DInsertion_rstree (HEAD *Head, RSNODE *node, RECT rect,
                    RSNODE **p_rchild, RECT *p_rect, int *inserted)
{
RSNODE  *p_b_node;
int     flag, promoted, ndx;
RECT    p_b_rect, brect;
/* Invoke choosesubtree with the level as a parameter       */
/* to find an appropriate node N in which to place the      */
/* new entry E                                              */
if (node == NULL) {
    p_rect->xlow = rect.xlow;
    p_rect->ylow = rect.ylow;
    p_rect->length_x = rect.length_x;
    p_rect->length_y = rect.length_y;
    (*p_rchild) = NULL;
    Head->total_rect++;
    return YES;
    }
/* Assuming that after deletion reinsertion of rect needs    */
/* to be checked for following just like in actual insertion */
/* select the right entry to be placed in node               */
/* choose subtree which is the sub function in choose leaf   */
/* check if the child pointers in N point to leaves          */
if (is_child_leaves (node))
    flag = determine_min_overlap_cost (rect, node, &ndx);
/* if the child pointers in N do not point to leaves */
else
    flag = select_entry_in_node (rect, node, &ndx);
/* checking for duplication errors */
if (flag && node->child_ptr[ndx] == NULL) {
    printf ("\nRectangle with same dimensions");
    return NO;
    }
/* recursively call and check if promotion occurs.               */
/* descend until leaf is reached  as said in choose leaf function */
```

```
/* set N to be the child node pointed to by F.p and repeat.       */
promoted = DInsertion_rstree (Head, node->child_ptr[ndx], rect,
                        &p_b_node, &p_b_rect, inserted);
if (node->child_ptr[ndx] != NULL) {
   brect = bounded_box (node->child_ptr[ndx]);
   node->rect[ndx].xlow = brect.xlow;
   node->rect[ndx].ylow = brect.ylow;
   node->rect[ndx].length_x = brect.length_x;
   node->rect[ndx].length_y = brect.length_y;
   }
if (!promoted)
   return NO;
/* if N has less than M entries accomodate E in N                 */
/* if N has M entries invoke Overflow Treatment                   */
/* with the level of N as a parameter for                         */
/* reinsertion or split                                           */
if (node->nofrect < MAXRECT) {
   put_rect_in_leaf (p_b_rect, p_b_node, node, ndx);
   *inserted = YES;
   return NO;
   }
else {  /* overflow treatment function is performed here         */
   /* if the level is not the root level and this is the first   */
   /* call of overflow treatment in the given level during the   */
   /* insertion of one data rectangle                            */
   if ((node->father != NULL) &&
       (first_call_overflow_trmt (Head, node))) {
      ReInsert (Head, &node, p_b_rect, p_b_node);
      }
   else {
      SPLIT_Q (p_b_rect, p_b_node, &node, p_rect,
            &(*p_rchild));
      Head->total_nodes++;
      *inserted = YES;
      return YES;
      }
   }  /* end of overflow treatment */
} /* end of DInsertion_rstree function */

/*********************************************************************
 reinsert rectangles in to the particular node and adjust pointers.
*********************************************************************/
void ReInsert (HEAD *Head, RSNODE **node, RECT rect, RSNODE *rchild)
{
int     i;
RECT    temp [MAXRECT + 1];
RSNODE *tempch [MAXRECT + 1];
struct rect_center {
       float   xcenter;
       float   ycenter;
       }  center [MAXRECT + 1], cbrect;
float  dist [MAXRECT + 1];
RECT   tmp;
/* for all M+1 entries of a node N compute the distance between   */
/* the centers of their rectangles and the center of the bounding */
/* rectangle of N                                                 */
for (i = 0; i < MAXRECT; i++) {
    temp[i].xlow = (*node)->rect[i].xlow;
    temp[i].ylow = (*node)->rect[i].ylow;
    temp[i].length_x = (*node)->rect[i].length_x;
    temp[i].length_y = (*node)->rect[i].length_y;
    tempch[i] = (*node)->child_ptr[i];
```

```
    center[i].xcenter = (temp[i].xlow + (temp[i].xlow + temp[i].length_x))
/ 2;
    center[i].ycenter = (temp[i].ylow + (temp[i].ylow + temp[i].length_y))
/ 2;
    }
temp[i].xlow = rect.xlow;
temp[i].ylow = rect.ylow;
temp[i].length_x = rect.length_x;
temp[i].length_y = rect.length_y;
tempch[i] = rchild;
center[i].xcenter = (temp[i].xlow + (temp[i].xlow + temp[i].length_x)) /
2;
center[i].ycenter = (temp[i].ylow + (temp[i].ylow + temp[i].length_y)) /
2;
for (i = 0; i < MAXRECT; i++)
    (*node)->child_ptr[i] = NULL;
/* calculate bounding box of N for finding distances */
bound_rect (temp[0], temp[1], &tmp);
for (i = 2; i < MAXRECT+1; i++)
    bound_rect (tmp, temp[i], &tmp);
cbrect.xcenter = (tmp.xlow + (tmp.xlow + tmp.length_x)) / 2;
cbrect.ycenter = (tmp.ylow + (tmp.ylow + tmp.length_y)) / 2;
/* calculate distances between centers of each rectangle and bound box */
for (i = 0; i < MAXRECT + 1; i++) {
    dist [i] =  sqrt ((fabs (center[i].xcenter-cbrect.xcenter)*
                       fabs (center[i].xcenter-cbrect.xcenter)) +
                      (fabs (center[i].ycenter-cbrect.ycenter)*
                       fabs (center[i].ycenter-cbrect.ycenter))
                     );
    }
sort_bubble (dist);     /* done in decreasing order */
(*node)->nofrect = 0;
/* remove the first p entries from N and put the rest in the node */
for (i = int(ceil(RIP * MAXRECT)); i < MAXRECT + 1; i++) {
    put_node (*node, temp[i], tempch[i], (*node)->nofrect);
    }
/* adjust the bounding rectangle of N */
bound_rect (temp[int(ceil(RIP * MAXRECT))],
            temp[int(ceil(RIP * MAXRECT))+1], &tmp);
for (i = int(ceil(RIP * MAXRECT)) + 2; i < MAXRECT + 1; i++) {
    bound_rect (tmp, temp[i], &tmp);
    }
/* in the sort defined above, starting with the maximum distance,      */
/* also called as far reinsert or minimum distance or closed reinsert, */
/* invoke Insert to reinsert the entries                               */
for (i = 0; i < int(ceil(RIP * MAXRECT)); i++)
    Insert_Data (Head, temp[i]);
} /* end of ReInsert function */

/*****************************************************************
 bubble sort a set of entries in an array
*****************************************************************/
void sort_bubble (float dist [])
{
int    i, j;
float  temp;
for (i = 0; i < MAXRECT; i++) {
    for (j = 1; j < MAXRECT + 1; j++) {
        if (dist [i] < dist[j]) {       /* decreasing order */
           temp = dist[i];
           dist[i] = dist[j];
           dist[j] = temp;
```

```
            }
         }
      }
} /* end of sort_bubble function */

/*************************************************************************
 function to check for first call of overflow treatment
*************************************************************************/
int first_call_overflow_trmt (HEAD *Head, RSNODE *node)
{
int  count = 0;
while (node->father != NULL)
      count++;
if (Head->level[count] == YES) {  /* meaning this is first call */
   Head->level[count] = NO;       /* change first call status hence */
   return YES;
   }
else
   return NO;
} /* end of first_call_overflow_trmt function */
```

APPENDIX B

DATA SET LISTING

The data set listed in this appendix consisted of a Very Large Scale Integrated (VLSI) design layout file that was converted into a text file. This text file consisted of numerous rectangles that enclose spatial records. Each rectangle or spatial object consisted of part of a VLSI circuit design. The system that was implemented performed operations such as searching and retrieval, insertion, and deletion on this data file.

Two VLSI data files were used to test the validity of the spatial database system. One of the VLSI data set is listed in this appendix. It consisted of 1937 rectangles. Of these 1464 rectangles were unique. The remaining 473 rectangles were duplicates which were discarded before insertion into the index structure. The rectangles that were extracted from this data file to insert into the tree structure consisted of lines which began with "rect" that contained four coordinates signifying the lower x and y coordinates of a rectangle, and the upper x and y coordinates. These rectangles were inserted into the index structure of the spatial database system. The remaining part of the data file contained information about the VLSI circuit design.

```
magic
tech scmos
timestamp 716173621
use nanf211 NO
timestamp 638648608
transform 1 0 88 0 1 452
box -3 -2 35 74
use nanf211 N1
timestamp 638648608
transform 1 0 8 0 1 452
box -3 -2 35 74
use invf101 IO
timestamp 638648608
transform 1 0 72 0 1 452
box -3 -2 19 74
use dfrf301 FF_0
timestamp 649080197
transform 1 0 304 0 1 716
box -3 -2 155 74
use dfrf301 FF_1
timestamp 649080197
transform 1 0 552 0 1 164
box -3 -2 155 74
use dfrf301 FF_2
timestamp 649080197
transform 1 0 456 0 1 48
box -3 -2 155 74
use dfrf301 FF_3
timestamp 649080197
transform 1 0 560 0 1 972
box -3 -2 155 74
use dfrf301 FF_4
timestamp 649080197
transform 1 0 104 0 1 164
box -3 -2 155 74
use dfrf301 FF_5
timestamp 649080197
transform 1 0 8 0 1 972
box -3 -2 155 74
use dfrf301 FF_6
timestamp 649080197
transform 1 0 408 0 1 832
box -3 -2 155 74
use dfrf301 FF_7
timestamp 649080197
transform 1 0 208 0 1 972
box -3 -2 155 74
use dfrf301 FF_8
timestamp 649080197
transform 1 0 616 0 1 452
box -3 -2 155 74
use dfrf301 FF_9
timestamp 649080197
transform 1 0 656 0 1 48
box -3 -2 155 74
use dfrf301 FF_10
timestamp 649080197
transform 1 0 704 0 1 164
box -3 -2 155 74
use dfrf301 FF_11
timestamp 649080197
transform 1 0 464 0 1 452
```

```
box -3 -2 155 74
use dfrf301 FF_12
timestamp 649080197
transform 1 0 360 0 1 312
box -3 -2 155 74
use dfrf301 FF_13
timestamp 649080197
transform 1 0 352 0 1 592
box -3 -2 155 74
use dfrf301 FF_14
timestamp 649080197
transform 1 0 152 0 1 452
box -3 -2 155 74
use dfrf301 FF_15
timestamp 649080197
transform 1 0 704 0 1 832
box -3 -2 155 74
use muxf201 M_0
timestamp 638648608
transform -1 0 304 0 1 716
box -3 -2 51 74
use muxf201 M_1
timestamp 638648608
transform -1 0 552 0 1 164
box -3 -2 51 74
use muxf201 M_2
timestamp 638648608
transform -1 0 456 0 1 48
box -3 -2 51 74
use muxf201 M_3
timestamp 638648608
transform 1 0 608 0 1 832
box -3 -2 51 74
use muxf201 M_4
timestamp 638648608
transform 1 0 256 0 1 164
box -3 -2 51 74
use muxf201 M_5
timestamp 638648608
transform 1 0 160 0 1 972
box -3 -2 51 74
use muxf201 M_6
timestamp 638648608
transform 1 0 560 0 1 832
box -3 -2 51 74
use muxf201 M_7
timestamp 638648608
transform -1 0 208 0 1 832
box -3 -2 51 74
use muxf201 M_8
timestamp 638648608
transform -1 0 608 0 1 312
box -3 -2 51 74
use muxf201 M_9
timestamp 638648608
transform -1 0 656 0 1 48
box -3 -2 51 74
use muxf201 M_10
timestamp 638648608
transform -1 0 856 0 1 48
box -3 -2 51 74
use muxf201 M_11
```

```
timestamp 638648608
transform -1 0 560 0 1 312
box -3 -2 51 74
use muxf201 M_12
timestamp 638648608
transform -1 0 360 0 1 312
box -3 -2 51 74
use muxf201 M_13
timestamp 638648608
transform -1 0 304 0 1 592
box -3 -2 51 74
use muxf201 M_14
timestamp 638648608
transform 1 0 208 0 1 592
box -3 -2 51 74
use muxf201 M_15
timestamp 638648608
transform -1 0 704 0 1 832
box -3 -2 51 74
use invf103 I1
timestamp 638648608
transform -1 0 152 0 1 452
box -3 -2 35 74
use dfrf301 FF_0.2
timestamp 649080197
transform 1 0 256 0 1 832
box -3 -2 155 74
use dfrf301 FF_1.3
timestamp 649080197
transform 1 0 160 0 1 312
box -3 -2 155 74
use dfrf301 FF_2.4
timestamp 649080197
transform 1 0 56 0 1 592
box -3 -2 155 74
use dfrf301 FF_3.5
timestamp 649080197
transform 1 0 8 0 1 312
box -3 -2 155 74
use dfrf301 FF_4.6
timestamp 649080197
transform 1 0 608 0 1 716
box -3 -2 155 74
use dfrf301 FF_5.7
timestamp 649080197
transform 1 0 56 0 1 716
box -3 -2 155 74
use dfrf301 FF_6.8
timestamp 649080197
transform 1 0 712 0 1 972
box -3 -2 155 74
use dfrf301 FF_7.9
timestamp 649080197
transform 1 0 8 0 1 832
box -3 -2 155 74
use dfrf301 FF_8.10
timestamp 649080197
transform 1 0 304 0 1 452
box -3 -2 155 74
use dfrf301 FF_9.11
timestamp 649080197
transform 1 0 352 0 1 164
```

```
box -3 -2 155 74
use dfrf301 FF_10.12
timestamp 649080197
transform 1 0 456 0 1 716
box -3 -2 155 74
use dfrf301 FF_11.13
timestamp 649080197
transform 1 0 8 0 1 48
box -3 -2 155 74
use dfrf301 FF_12.14
timestamp 649080197
transform 1 0 600 0 1 592
box -3 -2 155 74
use dfrf301 FF_13.15
timestamp 649080197
transform 1 0 160 0 1 48
box -3 -2 155 74
use dfrf301 FF_14.16
timestamp 649080197
transform 1 0 408 0 1 972
box -3 -2 155 74
use dfrf301 FF_15.17
timestamp 649080197
transform 1 0 656 0 1 312
box -3 -2 155 74
use muxf201 M_0.18
timestamp 638648608
transform -1 0 256 0 1 832
box -3 -2 51 74
use muxf201 M_1.19
timestamp 638648608
transform 1 0 304 0 1 164
box -3 -2 51 74
use muxf201 M_2.20
timestamp 638648608
transform -1 0 56 0 1 592
box -3 -2 51 74
use muxf201 M_3.21
timestamp 638648608
transform 1 0 56 0 1 164
box -3 -2 51 74
use muxf201 M_4.22
timestamp 638648608
transform -1 0 800 0 1 592
box -3 -2 51 74
use muxf201 M_5.23
timestamp 638648608
transform 1 0 208 0 1 716
box -3 -2 51 74
use muxf201 M_6.24
timestamp 638648608
transform -1 0 808 0 1 716
box -3 -2 51 74
use muxf201 M_7.25
timestamp 638648608
transform -1 0 56 0 1 716
box -3 -2 51 74
use muxf201 M_8.26
timestamp 638648608
transform -1 0 352 0 1 592
box -3 -2 51 74
use muxf201 M_9.27
```

```
timestamp 638648608
transform -1 0 408 0 1 48
box -3 -2 51 74
use muxf201 M_10.28
timestamp 638648608
transform -1 0 552 0 1 592
box -3 -2 51 74
use muxf201 M_11.29
timestamp 638648608
transform -1 0 56 0 1 164
box -3 -2 51 74
use muxf201 M_12.30
timestamp 638648608
transform -1 0 600 0 1 592
box -3 -2 51 74
use muxf201 M_13.31
timestamp 638648608
transform 1 0 312 0 1 48
box -3 -2 51 74
use muxf201 M_14.32
timestamp 638648608
transform -1 0 408 0 1 972
box -3 -2 51 74
use muxf201 M_15.33
timestamp 638648608
transform -1 0 656 0 1 312
box -3 -2 51 74
use invf103 I1.34
timestamp 638648608
transform 1 0 40 0 1 452
box -3 -2 35 74
<< metal1 >>
rect -24 908 -21 911
rect -24 676 -21 679
rect -24 1072 -21 1075
rect -24 412 -21 415
rect -24 280 -21 283
rect -24 264 -21 267
rect -24 272 -21 275
rect -24 948 -21 951
rect -24 1080 -21 1083
rect -24 1048 -21 1051
rect -24 708 -21 711
rect -24 24 -21 27
rect -24 536 -21 539
rect -24 388 -21 391
rect -24 544 -21 547
rect -24 8 -21 11
rect -24 956 -21 959
rect -24 576 -21 579
rect 893 800 896 803
rect 893 264 896 267
rect 893 16 896 19
rect 893 956 896 959
rect 893 404 896 407
rect 893 1048 896 1051
rect 893 908 896 911
rect 893 1056 896 1059
rect 893 156 896 159
rect 893 24 896 27
rect 893 0 896 3
rect 893 428 896 431
```

```
rect 893 296 896 299
rect 893 692 896 695
rect 893 568 896 571
rect 893 1080 896 1083
rect -24 40 -21 43
rect -24 124 -21 127
rect 893 1064 896 1067
rect 893 256 896 259
rect 893 668 896 671
rect 893 248 896 251
rect 893 816 896 819
rect 893 924 896 927
rect 893 916 896 919
rect 893 792 896 795
rect 893 560 896 563
rect 893 8 896 11
rect 893 808 896 811
rect 893 40 896 43
rect 893 584 896 587
rect 893 32 896 35
rect 893 1072 896 1075
rect 893 388 896 391
rect -24 109 -16 117
rect -24 51 -16 59
rect 888 51 896 59
rect 888 109 896 117
rect -24 167 -16 175
rect 888 167 896 175
rect -24 225 -16 233
rect 888 225 896 233
rect -24 315 -16 323
rect 888 315 896 323
rect -24 373 -16 381
rect 888 373 896 381
rect -24 455 -16 463
rect 888 455 896 463
rect -24 513 -16 521
rect 888 513 896 521
rect -24 595 -16 603
rect 888 595 896 603
rect -24 653 -16 661
rect 888 653 896 661
rect -24 719 -16 727
rect 888 719 896 727
rect -24 777 -16 785
rect 888 777 896 785
rect -24 835 -16 843
rect 888 835 896 843
rect -24 893 -16 901
rect 888 893 896 901
rect -24 975 -16 983
rect 888 975 896 983
rect -24 1033 -16 1041
rect 888 1033 896 1041
rect 850 1083 854 1084
rect 722 1083 726 1084
rect 570 1083 574 1084
rect 562 1083 566 1084
rect 850 1080 896 1083
rect 570 1080 726 1083
rect -24 1080 566 1083
rect 554 1075 558 1076
```

```
rect 434 1075 438 1076
rect 402 1075 406 1076
rect 226 1075 230 1076
rect 554 1072 896 1075
rect 402 1072 438 1075
rect -24 1072 230 1075
rect 394 1067 398 1068
rect 362 1067 366 1068
rect 210 1067 214 1068
rect 394 1064 896 1067
rect 210 1064 366 1067
rect 354 1059 358 1060
rect 338 1059 342 1060
rect 138 1059 142 1060
rect 354 1056 896 1059
rect 138 1056 342 1059
rect 178 1051 182 1052
rect 146 1051 150 1052
rect 178 1048 896 1051
rect -24 1048 150 1051
rect -24 1033 896 1041
rect -24 975 896 983
rect 706 964 710 968
rect 586 967 590 968
rect 570 967 574 968
rect 418 967 422 968
rect 218 967 222 968
rect 18 967 22 968
rect 586 964 613 967
rect 18 964 574 967
rect 706 959 709 964
rect 610 960 613 964
rect 626 959 630 960
rect 610 956 614 960
rect 554 959 558 960
rect 386 959 390 960
rect 378 959 382 960
rect 242 959 246 960
rect 626 956 896 959
rect 386 956 558 959
rect -24 956 382 959
rect 850 948 854 952
rect 730 951 734 952
rect 698 951 702 952
rect 602 951 606 952
rect 218 951 222 952
rect 202 951 206 952
rect 194 951 198 952
rect 186 951 190 952
rect 698 948 734 951
rect 202 948 653 951
rect -24 948 198 951
rect 650 944 653 948
rect 850 943 854 944
rect 682 943 686 944
rect 650 940 654 944
rect 634 943 638 944
rect 170 943 174 944
rect 162 943 166 944
rect 34 943 38 944
rect 682 940 854 943
rect 170 940 638 943
```

```
rect 34 940 166 943
rect 850 936 853 940
rect 850 932 854 936
rect 842 935 846 936
rect 834 935 838 936
rect 690 935 694 936
rect 538 935 542 936
rect 338 935 342 936
rect 282 935 286 936
rect 250 935 254 936
rect 234 935 238 936
rect 178 935 182 936
rect 154 935 158 936
rect 146 932 150 936
rect 338 932 846 935
rect 250 932 286 935
rect 202 932 238 935
rect 154 932 182 935
rect 202 928 205 932
rect 146 912 149 932
rect 274 927 278 928
rect 258 927 262 928
rect 210 927 214 928
rect 202 924 206 928
rect 274 924 896 927
rect 210 924 262 927
rect 858 919 862 920
rect 842 919 846 920
rect 738 919 742 920
rect 722 919 726 920
rect 690 919 694 920
rect 586 919 590 920
rect 562 919 566 920
rect 394 916 398 920
rect 354 919 358 920
rect 842 916 896 919
rect 722 916 742 919
rect 562 916 694 919
rect 186 916 358 919
rect 394 911 397 916
rect 186 912 189 916
rect 578 911 582 912
rect 562 911 566 912
rect 434 911 438 912
rect 402 911 406 912
rect 234 911 238 912
rect 186 908 190 912
rect 178 911 182 912
rect 146 911 150 912
rect 26 911 30 912
rect 578 908 896 911
rect 434 908 566 911
rect 234 908 406 911
rect 146 908 182 911
rect -24 908 30 911
rect -24 893 896 901
rect -24 835 896 843
rect 842 827 846 828
rect 786 827 790 828
rect 778 827 782 828
rect 690 827 694 828
rect 674 824 678 828
```

```
rect 578 827 582 828
rect 554 827 558 828
rect 538 824 542 828
rect 466 827 470 828
rect 418 827 422 828
rect 266 827 270 828
rect 234 827 238 828
rect 194 827 198 828
rect 170 824 174 828
rect 18 827 22 828
rect 786 824 846 827
rect 690 824 782 827
rect 554 824 582 827
rect 266 824 470 827
rect 194 824 238 827
rect 18 824 69 827
rect 674 819 677 824
rect 538 819 541 824
rect 170 819 173 824
rect 66 812 69 824
rect 754 819 758 820
rect 746 819 750 820
rect 722 819 726 820
rect 626 819 630 820
rect 586 819 590 820
rect 386 819 390 820
rect 274 816 278 820
rect 266 819 270 820
rect 226 819 230 820
rect 218 816 222 820
rect 34 819 38 820
rect 754 816 896 819
rect 722 816 750 819
rect 626 816 677 819
rect 386 816 590 819
rect 226 816 270 819
rect 74 816 173 819
rect 34 816 53 819
rect 434 812 437 816
rect 274 811 277 816
rect 218 803 221 816
rect 74 812 77 816
rect 50 812 53 816
rect 602 811 606 812
rect 594 811 598 812
rect 482 811 486 812
rect 434 808 438 812
rect 330 811 334 812
rect 298 811 302 812
rect 226 811 230 812
rect 210 811 214 812
rect 82 811 86 812
rect 74 808 78 812
rect 66 808 70 812
rect 50 808 54 812
rect 602 808 896 811
rect 482 808 598 811
rect 298 808 334 811
rect 226 808 277 811
rect 82 808 214 811
rect 450 803 454 804
rect 282 803 286 804
```

```
rect 258 803 262 804
rect 162 803 166 804
rect 154 800 158 804
rect 146 803 150 804
rect 282 800 896 803
rect 162 800 262 803
rect 26 800 150 803
rect 154 795 157 800
rect 26 796 29 800
rect 34 795 38 796
rect 26 792 30 796
rect 34 792 896 795
rect -24 777 896 785
rect -24 719 896 727
rect 802 711 806 712
rect 746 711 750 712
rect 738 711 742 712
rect 730 711 734 712
rect 586 711 590 712
rect 570 711 574 712
rect 338 711 342 712
rect 322 711 326 712
rect 746 708 806 711
rect 586 708 742 711
rect 338 708 574 711
rect -24 708 326 711
rect 794 703 798 704
rect 634 703 638 704
rect 602 703 606 704
rect 530 703 534 704
rect 378 703 382 704
rect 298 703 302 704
rect 266 703 270 704
rect 250 703 254 704
rect 226 703 230 704
rect 202 703 206 704
rect 10 700 14 704
rect 634 700 798 703
rect 530 700 606 703
rect 298 700 382 703
rect 250 700 270 703
rect 202 700 230 703
rect 10 688 13 700
rect 282 695 286 696
rect 282 692 896 695
rect 498 680 501 692
rect 754 687 758 688
rect 594 687 598 688
rect 546 687 550 688
rect 362 687 366 688
rect 314 687 318 688
rect 306 687 310 688
rect 266 687 270 688
rect 10 687 14 688
rect 754 684 781 687
rect 546 684 598 687
rect 314 684 366 687
rect 10 684 310 687
rect 778 680 781 684
rect 778 676 782 680
rect 746 679 750 680
rect 578 679 582 680
```

```
rect 554 679 558 680
rect 506 679 510 680
rect 498 676 502 680
rect 482 679 486 680
rect 186 679 190 680
rect 82 679 86 680
rect 50 679 54 680
rect 42 679 46 680
rect 578 676 750 679
rect 506 676 558 679
rect 186 676 486 679
rect 50 676 86 679
rect -24 676 46 679
rect 202 671 206 672
rect 34 671 38 672
rect 34 668 896 671
rect -24 653 896 661
rect -24 595 896 603
rect 746 587 750 588
rect 626 587 630 588
rect 594 587 598 588
rect 506 587 510 588
rect 306 587 310 588
rect 170 587 174 588
rect 26 587 30 588
rect 746 584 896 587
rect 594 584 630 587
rect 306 584 510 587
rect 26 584 174 587
rect 634 579 638 580
rect 618 579 622 580
rect -24 576 638 579
rect 226 571 230 572
rect 210 571 214 572
rect 178 571 182 572
rect 74 568 78 572
rect 42 571 46 572
rect 10 568 14 572
rect 226 568 896 571
rect 178 568 214 571
rect 26 568 46 571
rect 298 556 301 568
rect 74 563 77 568
rect 26 564 29 568
rect 10 555 13 568
rect 330 563 334 564
rect 322 563 326 564
rect 250 563 254 564
rect 26 560 30 564
rect 330 560 896 563
rect 306 560 326 563
rect 146 560 254 563
rect 74 560 133 563
rect 450 548 453 560
rect 306 556 309 560
rect 146 556 149 560
rect 130 556 133 560
rect 770 552 774 556
rect 754 555 758 556
rect 554 555 558 556
rect 482 552 486 556
rect 338 552 342 556
```

```
rect 306 552 310 556
rect 298 552 302 556
rect 146 552 150 556
rect 130 552 134 556
rect 42 555 46 556
rect 554 552 758 555
rect 10 552 46 555
rect 770 531 773 552
rect 618 540 621 552
rect 338 547 341 552
rect 730 544 734 548
rect 610 544 614 548
rect 586 547 590 548
rect 482 547 486 548
rect 450 544 454 548
rect 154 547 158 548
rect 482 544 597 547
rect -24 544 341 547
rect 730 531 733 544
rect 610 531 613 544
rect 594 540 597 544
rect 618 536 622 540
rect 594 536 598 540
rect 522 539 526 540
rect 466 539 470 540
rect -24 536 526 539
rect 754 531 758 532
rect 746 531 750 532
rect 626 531 630 532
rect 362 531 366 532
rect 346 531 350 532
rect 330 531 334 532
rect 322 531 326 532
rect 274 531 278 532
rect 754 528 773 531
rect 730 528 750 531
rect 362 528 630 531
rect 330 528 350 531
rect 274 528 326 531
rect -24 513 896 521
rect 458 489 462 513
rect 458 463 462 479
rect -24 455 896 463
rect 746 444 750 448
rect 642 444 646 448
rect 626 447 630 448
rect 474 447 478 448
rect 466 444 470 448
rect 314 447 318 448
rect 178 447 182 448
rect 162 447 166 448
rect 58 447 62 448
rect 10 444 14 448
rect 474 444 630 447
rect 18 444 318 447
rect 746 439 749 444
rect 642 439 645 444
rect 466 431 469 444
rect 18 440 21 444
rect 10 431 13 444
rect 786 439 790 440
rect 602 439 606 440
```

```
rect 554 439 558 440
rect 490 439 494 440
rect 314 439 318 440
rect 146 439 150 440
rect 138 439 142 440
rect 114 439 118 440
rect 106 436 110 440
rect 90 439 94 440
rect 26 439 30 440
rect 18 436 22 440
rect 746 436 790 439
rect 602 436 645 439
rect 490 436 558 439
rect 146 436 318 439
rect 114 436 142 439
rect 26 436 94 439
rect 106 431 109 436
rect 610 431 614 432
rect 538 431 542 432
rect 522 431 526 432
rect 386 431 390 432
rect 354 431 358 432
rect 330 431 334 432
rect 154 431 158 432
rect 82 431 86 432
rect 74 431 78 432
rect 538 428 896 431
rect 466 428 526 431
rect 354 428 390 431
rect 154 428 334 431
rect 82 428 109 431
rect 10 428 78 431
rect 762 423 766 424
rect 586 423 590 424
rect 578 423 582 424
rect 306 423 310 424
rect 178 423 182 424
rect 162 423 166 424
rect 50 423 54 424
rect 34 423 38 424
rect 586 420 766 423
rect 306 420 582 423
rect 162 420 182 423
rect 34 420 54 423
rect 754 415 758 416
rect 570 415 574 416
rect 546 415 550 416
rect 570 412 758 415
rect -24 412 550 415
rect 298 407 302 408
rect 282 404 286 408
rect 130 404 134 408
rect 298 404 896 407
rect 282 399 285 404
rect 130 391 133 404
rect 682 399 686 400
rect 650 399 654 400
rect 618 396 622 400
rect 562 399 566 400
rect 514 399 518 400
rect 490 399 494 400
rect 482 399 486 400
```

```
rect 434 399 438 400
rect 290 399 294 400
rect 650 396 686 399
rect 514 396 566 399
rect 282 396 494 399
rect 618 391 621 396
rect 802 391 806 392
rect 634 391 638 392
rect 610 391 614 392
rect 162 391 166 392
rect 146 391 150 392
rect 26 391 30 392
rect 634 388 896 391
rect 162 388 621 391
rect 130 388 150 391
rect -24 388 30 391
rect -24 373 896 381
rect -24 315 896 323
rect 522 307 526 308
rect 498 307 502 308
rect 338 304 342 308
rect 314 304 318 308
rect 298 307 302 308
rect 282 307 286 308
rect 34 304 38 308
rect 26 304 30 308
rect 498 304 526 307
rect 282 304 302 307
rect 338 299 341 304
rect 314 299 317 304
rect 34 299 37 304
rect 26 292 29 304
rect 506 299 510 300
rect 298 299 302 300
rect 58 299 62 300
rect 338 296 896 299
rect 298 296 317 299
rect 34 296 62 299
rect 546 288 550 292
rect 530 291 534 292
rect 26 291 30 292
rect 26 288 534 291
rect 546 283 549 288
rect 682 283 686 284
rect 522 283 526 284
rect 514 283 518 284
rect 506 283 510 284
rect 370 283 374 284
rect 282 283 286 284
rect 274 283 278 284
rect 250 283 254 284
rect 178 283 182 284
rect 42 283 46 284
rect 570 280 837 283
rect 522 280 549 283
rect 370 280 518 283
rect 250 280 286 283
rect -24 280 182 283
rect 834 276 837 280
rect 570 276 573 280
rect 834 272 838 276
rect 722 275 726 276
```

```
rect 586 275 590 276
rect 570 272 574 276
rect 562 275 566 276
rect 282 275 286 276
rect 586 272 726 275
rect -24 272 566 275
rect 698 267 702 268
rect 530 267 534 268
rect 514 267 518 268
rect 498 267 502 268
rect 378 267 382 268
rect 362 267 366 268
rect 242 267 246 268
rect 146 267 150 268
rect 82 267 86 268
rect 530 264 896 267
rect 498 264 518 267
rect 242 264 382 267
rect -24 264 150 267
rect 322 259 326 260
rect 306 259 310 260
rect 290 259 294 260
rect 234 259 238 260
rect 138 259 142 260
rect 122 259 126 260
rect 114 259 118 260
rect 18 259 22 260
rect 306 256 896 259
rect 122 256 294 259
rect 18 256 118 259
rect 18 252 21 256
rect 154 251 158 252
rect 74 251 78 252
rect 18 248 22 252
rect 74 248 896 251
rect 842 243 846 244
rect 730 243 734 244
rect 690 243 694 244
rect 666 243 670 244
rect 562 243 566 244
rect 490 243 494 244
rect 306 243 310 244
rect 186 243 190 244
rect 162 243 166 244
rect 98 243 102 244
rect 10 243 14 244
rect 730 240 846 243
rect 490 240 694 243
rect 186 240 310 243
rect 10 240 166 243
rect -24 225 896 233
rect -24 167 896 175
rect 722 159 726 160
rect 714 159 718 160
rect 690 159 694 160
rect 666 159 670 160
rect 410 159 414 160
rect 370 159 374 160
rect 298 159 302 160
rect 258 159 262 160
rect 130 159 134 160
rect 114 156 118 160
```

```
rect 98 156 102 160
rect 42 156 46 160
rect 722 156 896 159
rect 666 156 718 159
rect 298 156 613 159
rect 130 156 262 159
rect 610 152 613 156
rect 114 151 117 156
rect 98 143 101 156
rect 42 151 45 156
rect 850 151 854 152
rect 682 151 686 152
rect 650 151 654 152
rect 610 148 614 152
rect 578 151 582 152
rect 546 151 550 152
rect 522 151 526 152
rect 330 151 334 152
rect 314 151 318 152
rect 298 151 302 152
rect 242 151 246 152
rect 834 148 854 151
rect 650 148 686 151
rect 546 148 582 151
rect 330 148 526 151
rect 298 148 318 151
rect 114 148 246 151
rect 26 148 45 151
rect 834 128 837 148
rect 26 136 29 148
rect 514 143 518 144
rect 490 143 494 144
rect 362 143 366 144
rect 346 143 350 144
rect 34 140 38 144
rect 514 140 829 143
rect 362 140 494 143
rect 98 140 357 143
rect 826 128 829 140
rect 466 136 469 140
rect 354 128 357 140
rect 34 135 37 140
rect 786 135 790 136
rect 570 135 574 136
rect 482 135 486 136
rect 466 132 470 136
rect 402 135 406 136
rect 378 135 382 136
rect 26 132 30 136
rect 482 132 790 135
rect 378 132 406 135
rect 34 132 181 135
rect 586 128 589 132
rect 178 128 181 132
rect 154 128 157 132
rect 834 124 838 128
rect 826 124 830 128
rect 802 127 806 128
rect 634 127 638 128
rect 586 124 590 128
rect 498 127 502 128
rect 474 127 478 128
```

```
rect 386 127 390 128
rect 354 124 358 128
rect 314 127 318 128
rect 186 127 190 128
rect 178 124 182 128
rect 154 124 158 128
rect 138 127 142 128
rect 122 127 126 128
rect 634 124 806 127
rect 386 124 502 127
rect 186 124 318 127
rect -24 124 142 127
rect -24 109 896 117
rect -24 51 896 59
rect 178 43 182 44
rect 170 43 174 44
rect 18 43 22 44
rect 178 40 896 43
rect -24 40 174 43
rect 330 35 334 36
rect 306 35 310 36
rect 306 32 896 35
rect 802 27 806 28
rect 626 27 630 28
rect 378 27 382 28
rect 802 24 896 27
rect -24 24 630 27
rect 602 19 606 20
rect 434 19 438 20
rect 426 19 430 20
rect 26 19 30 20
rect 434 16 896 19
rect 26 16 430 19
rect 474 11 478 12
rect 450 8 454 12
rect 338 11 342 12
rect 298 11 302 12
rect 474 8 896 11
rect -24 8 342 11
rect 450 3 453 8
rect 834 3 838 4
rect 810 3 814 4
rect 610 3 614 4
rect 482 3 486 4
rect 834 0 896 3
rect 610 0 814 3
rect 450 0 486 3
<< metal2 >>
rect 850 1080 854 1084
rect 722 1080 726 1084
rect 570 1080 574 1084
rect 562 1080 566 1084
rect 850 952 853 1080
rect 722 967 725 1080
rect 570 968 573 1080
rect 562 920 565 1080
rect 554 1072 558 1076
rect 434 1072 438 1076
rect 402 1072 406 1076
rect 226 1072 230 1076
rect 554 960 557 1072
rect 434 998 437 1072
```

```
rect 402 1007 405 1072
rect 226 820 229 1072
rect 394 1064 398 1068
rect 362 1064 366 1068
rect 210 1064 214 1068
rect 394 920 397 1064
rect 362 1003 365 1064
rect 210 928 213 1064
rect 354 1056 358 1060
rect 338 1056 342 1060
rect 138 1056 142 1060
rect 354 920 357 1056
rect 338 936 341 1056
rect 138 859 141 1056
rect 178 1048 182 1052
rect 146 1048 150 1052
rect 178 936 181 1048
rect 146 936 149 1048
rect 162 944 165 1007
rect 858 920 861 1003
rect 706 968 709 1003
rect 202 952 205 1003
rect 154 936 157 1003
rect 378 960 381 1002
rect 186 952 189 1002
rect 842 936 845 999
rect 690 936 693 999
rect 538 936 541 999
rect 386 960 389 999
rect 738 920 741 998
rect 586 968 589 998
rect 234 936 237 998
rect 34 944 37 998
rect 418 968 421 996
rect 218 968 221 996
rect 18 968 21 996
rect 706 964 710 968
rect 586 964 590 968
rect 570 964 574 968
rect 418 964 422 968
rect 218 964 222 968
rect 18 964 22 968
rect 714 964 725 967
rect 714 856 717 964
rect 418 828 421 964
rect 18 828 21 964
rect 626 956 630 960
rect 610 956 614 960
rect 554 956 558 960
rect 386 956 390 960
rect 378 956 382 960
rect 242 956 246 960
rect 626 859 629 956
rect 610 867 613 956
rect 242 695 245 956
rect 850 948 854 952
rect 730 948 734 952
rect 698 948 702 952
rect 602 948 606 952
rect 218 948 222 952
rect 202 948 206 952
rect 194 948 198 952
```

```
rect 186 948 190 952
rect 850 944 853 948
rect 730 858 733 948
rect 698 867 701 948
rect 602 863 605 948
rect 218 820 221 948
rect 194 828 197 948
rect 850 940 854 944
rect 682 940 686 944
rect 650 940 654 944
rect 634 940 638 944
rect 170 940 174 944
rect 162 940 166 944
rect 34 940 38 944
rect 850 936 853 940
rect 682 859 685 940
rect 650 911 653 940
rect 634 862 637 940
rect 170 828 173 940
rect 850 932 854 936
rect 842 935 846 936
rect 834 935 838 936
rect 690 932 694 936
rect 538 932 542 936
rect 338 932 342 936
rect 282 932 286 936
rect 250 932 254 936
rect 234 932 238 936
rect 178 932 182 936
rect 154 932 158 936
rect 146 932 150 936
rect 834 932 846 935
rect 850 863 853 932
rect 834 859 837 932
rect 538 828 541 932
rect 282 858 285 932
rect 250 867 253 932
rect 274 924 278 928
rect 258 924 262 928
rect 210 924 214 928
rect 202 924 206 928
rect 274 820 277 924
rect 258 827 261 924
rect 210 863 213 924
rect 202 867 205 924
rect 858 916 862 920
rect 842 916 846 920
rect 738 916 742 920
rect 722 916 726 920
rect 690 916 694 920
rect 586 916 590 920
rect 562 916 566 920
rect 394 916 398 920
rect 354 916 358 920
rect 842 828 845 916
rect 722 820 725 916
rect 690 828 693 916
rect 586 862 589 916
rect 578 908 582 912
rect 562 908 566 912
rect 434 908 438 912
rect 402 908 406 912
```

```
rect 234 908 238 912
rect 186 908 190 912
rect 178 908 182 912
rect 146 908 150 912
rect 26 908 30 912
rect 650 908 661 911
rect 658 863 661 908
rect 650 863 653 908
rect 578 828 581 908
rect 562 867 565 908
rect 434 858 437 908
rect 402 863 405 908
rect 234 859 237 908
rect 186 859 189 908
rect 178 862 181 908
rect 146 804 149 908
rect 26 811 29 908
rect 554 828 557 863
rect 162 804 165 863
rect 154 804 157 863
rect 674 828 677 862
rect 386 820 389 859
rect 34 820 37 858
rect 266 828 269 856
rect 842 824 846 828
rect 786 824 790 828
rect 778 824 782 828
rect 690 824 694 828
rect 674 824 678 828
rect 578 824 582 828
rect 554 824 558 828
rect 538 824 542 828
rect 466 824 470 828
rect 418 824 422 828
rect 266 824 270 828
rect 234 824 238 828
rect 194 824 198 828
rect 170 824 174 828
rect 18 824 22 828
rect 250 824 261 827
rect 786 743 789 824
rect 778 746 781 824
rect 466 740 469 824
rect 250 704 253 824
rect 234 746 237 824
rect 754 816 758 820
rect 746 816 750 820
rect 722 816 726 820
rect 626 816 630 820
rect 586 816 590 820
rect 386 816 390 820
rect 274 816 278 820
rect 266 816 270 820
rect 226 816 230 820
rect 218 816 222 820
rect 34 816 38 820
rect 754 688 757 816
rect 746 712 749 816
rect 626 687 629 816
rect 586 712 589 816
rect 266 803 269 816
rect 602 808 606 812
```

```
rect 594 808 598 812
rect 482 808 486 812
rect 434 808 438 812
rect 330 808 334 812
rect 298 808 302 812
rect 226 808 230 812
rect 210 808 214 812
rect 82 808 86 812
rect 74 808 78 812
rect 66 808 70 812
rect 50 808 54 812
rect 18 808 29 811
rect 602 704 605 808
rect 594 688 597 808
rect 482 742 485 808
rect 434 743 437 808
rect 330 742 333 808
rect 298 751 301 808
rect 226 704 229 808
rect 210 751 213 808
rect 82 742 85 808
rect 74 572 77 808
rect 66 563 69 808
rect 50 751 53 808
rect 18 531 21 808
rect 450 800 454 804
rect 282 800 286 804
rect 258 800 262 804
rect 162 800 166 804
rect 154 800 158 804
rect 146 800 150 804
rect 266 800 277 803
rect 450 747 453 800
rect 282 743 285 800
rect 274 746 277 800
rect 258 671 261 800
rect 34 792 38 796
rect 26 792 30 796
rect 34 743 37 792
rect 26 746 29 792
rect 802 712 805 751
rect 762 679 765 747
rect 202 704 205 747
rect 10 704 13 747
rect 738 712 741 743
rect 186 680 189 743
rect 634 704 637 742
rect 618 703 621 740
rect 314 688 317 740
rect 802 708 806 712
rect 746 708 750 712
rect 738 711 742 712
rect 730 711 734 712
rect 586 708 590 712
rect 570 708 574 712
rect 338 708 342 712
rect 322 708 326 712
rect 730 708 742 711
rect 730 548 733 708
rect 586 548 589 708
rect 570 622 573 708
rect 338 556 341 708
```

```
rect 322 564 325 708
rect 794 700 798 704
rect 634 700 638 704
rect 602 700 606 704
rect 530 700 534 704
rect 378 700 382 704
rect 298 700 302 704
rect 266 700 270 704
rect 250 700 254 704
rect 226 700 230 704
rect 202 700 206 704
rect 10 700 14 704
rect 610 700 621 703
rect 794 627 797 700
rect 610 548 613 700
rect 530 619 533 700
rect 378 618 381 700
rect 298 627 301 700
rect 266 688 269 700
rect 282 692 286 696
rect 234 692 245 695
rect 282 619 285 692
rect 234 622 237 692
rect 754 684 758 688
rect 594 684 598 688
rect 546 684 550 688
rect 362 684 366 688
rect 314 684 318 688
rect 306 684 310 688
rect 266 684 270 688
rect 10 684 14 688
rect 618 684 629 687
rect 618 580 621 684
rect 546 627 549 684
rect 362 532 365 684
rect 314 448 317 684
rect 306 588 309 684
rect 10 572 13 684
rect 778 676 782 680
rect 746 676 750 680
rect 578 676 582 680
rect 554 676 558 680
rect 506 676 510 680
rect 498 676 502 680
rect 482 676 486 680
rect 186 676 190 680
rect 82 676 86 680
rect 50 676 54 680
rect 42 676 46 680
rect 754 676 765 679
rect 778 619 781 676
rect 754 556 757 676
rect 746 588 749 676
rect 578 619 581 676
rect 554 556 557 676
rect 506 588 509 676
rect 498 623 501 676
rect 482 556 485 676
rect 186 619 189 676
rect 82 618 85 676
rect 50 627 53 676
rect 42 572 45 676
```

```
rect 202 668 206 672
rect 34 668 38 672
rect 250 668 261 671
rect 258 623 261 668
rect 250 564 253 668
rect 202 623 205 668
rect 34 619 37 668
rect 594 588 597 627
rect 346 532 349 627
rect 210 572 213 627
rect 770 556 773 622
rect 522 540 525 622
rect 274 532 277 622
rect 26 588 29 622
rect 330 564 333 619
rect 226 572 229 619
rect 626 588 629 618
rect 746 584 750 588
rect 626 584 630 588
rect 594 584 598 588
rect 506 584 510 588
rect 306 584 310 588
rect 170 584 174 588
rect 26 584 30 588
rect 170 439 173 584
rect 634 576 638 580
rect 618 576 622 580
rect 634 399 637 576
rect 226 568 230 572
rect 210 568 214 572
rect 178 568 182 572
rect 74 568 78 572
rect 42 568 46 572
rect 10 568 14 572
rect 178 478 181 568
rect 330 560 334 564
rect 322 560 326 564
rect 250 560 254 564
rect 26 560 30 564
rect 58 560 69 563
rect 58 448 61 560
rect 26 440 29 560
rect 770 552 774 556
rect 754 552 758 556
rect 554 552 558 556
rect 482 552 486 556
rect 338 552 342 556
rect 306 552 310 556
rect 298 552 302 556
rect 146 552 150 556
rect 130 552 134 556
rect 42 552 46 556
rect 482 548 485 552
rect 306 424 309 552
rect 298 483 301 552
rect 146 440 149 552
rect 130 408 133 552
rect 42 483 45 552
rect 730 544 734 548
rect 610 544 614 548
rect 586 544 590 548
rect 482 544 486 548
```

```
rect 450 544 454 548
rect 154 544 158 548
rect 482 400 485 544
rect 450 483 453 544
rect 154 432 157 544
rect 618 536 622 540
rect 594 536 598 540
rect 522 536 526 540
rect 466 536 470 540
rect 618 400 621 536
rect 594 479 597 536
rect 466 448 469 536
rect 754 528 758 532
rect 746 528 750 532
rect 626 528 630 532
rect 362 528 366 532
rect 346 528 350 532
rect 330 528 334 532
rect 322 528 326 532
rect 274 528 278 532
rect 10 528 21 531
rect 754 416 757 528
rect 746 448 749 528
rect 626 448 629 528
rect 330 478 333 528
rect 322 283 325 528
rect 10 448 13 528
rect 106 440 109 492
rect 458 480 462 488
rect 114 440 117 486
rect 90 440 93 486
rect 34 424 37 486
rect 762 424 765 483
rect 610 432 613 483
rect 138 440 141 483
rect 82 432 85 483
rect 74 432 77 483
rect 50 424 53 483
rect 434 400 437 479
rect 282 408 285 479
rect 642 448 645 478
rect 490 440 493 478
rect 474 448 477 476
rect 162 448 165 476
rect 746 444 750 448
rect 642 444 646 448
rect 626 444 630 448
rect 474 444 478 448
rect 466 444 470 448
rect 314 444 318 448
rect 178 444 182 448
rect 162 444 166 448
rect 58 444 62 448
rect 10 444 14 448
rect 178 431 181 444
rect 786 436 790 440
rect 602 436 606 440
rect 554 436 558 440
rect 490 436 494 440
rect 314 436 318 440
rect 146 436 150 440
rect 138 436 142 440
```

```
rect 114 436 118 440
rect 106 436 110 440
rect 90 436 94 440
rect 26 436 30 440
rect 18 436 22 440
rect 162 436 173 439
rect 786 339 789 436
rect 602 347 605 436
rect 554 347 557 436
rect 314 308 317 436
rect 162 424 165 436
rect 18 260 21 436
rect 610 428 614 432
rect 538 428 542 432
rect 522 428 526 432
rect 386 428 390 432
rect 354 428 358 432
rect 330 428 334 432
rect 154 428 158 432
rect 82 428 86 432
rect 74 428 78 432
rect 170 428 181 431
rect 538 339 541 428
rect 522 308 525 428
rect 386 338 389 428
rect 354 347 357 428
rect 330 342 333 428
rect 170 336 173 428
rect 762 420 766 424
rect 586 420 590 424
rect 578 420 582 424
rect 306 420 310 424
rect 178 420 182 424
rect 162 420 166 424
rect 50 420 54 424
rect 34 420 38 424
rect 586 276 589 420
rect 578 342 581 420
rect 178 284 181 420
rect 754 412 758 416
rect 570 412 574 416
rect 546 412 550 416
rect 570 291 573 412
rect 546 292 549 412
rect 298 404 302 408
rect 282 404 286 408
rect 130 404 134 408
rect 298 308 301 404
rect 682 396 686 400
rect 650 396 654 400
rect 618 396 622 400
rect 562 396 566 400
rect 514 396 518 400
rect 490 399 494 400
rect 482 399 486 400
rect 434 396 438 400
rect 290 396 294 400
rect 626 396 637 399
rect 482 396 494 399
rect 682 338 685 396
rect 650 347 653 396
rect 626 342 629 396
```

```
rect 562 343 565 396
rect 514 284 517 396
rect 490 267 493 396
rect 290 260 293 396
rect 802 388 806 392
rect 634 388 638 392
rect 610 388 614 392
rect 162 388 166 392
rect 146 388 150 392
rect 26 388 30 392
rect 802 343 805 388
rect 634 339 637 388
rect 610 343 613 388
rect 162 244 165 388
rect 146 268 149 388
rect 26 308 29 388
rect 506 300 509 343
rect 306 260 309 343
rect 154 252 157 343
rect 530 292 533 342
rect 338 308 341 339
rect 138 260 141 339
rect 186 244 189 338
rect 34 308 37 338
rect 666 244 669 336
rect 370 291 373 336
rect 522 304 526 308
rect 498 304 502 308
rect 338 304 342 308
rect 314 304 318 308
rect 298 304 302 308
rect 282 304 286 308
rect 34 304 38 308
rect 26 304 30 308
rect 498 268 501 304
rect 282 284 285 304
rect 506 296 510 300
rect 298 296 302 300
rect 58 296 62 300
rect 298 160 301 296
rect 58 199 61 296
rect 546 288 550 292
rect 530 288 534 292
rect 26 288 30 292
rect 562 288 573 291
rect 370 288 381 291
rect 562 276 565 288
rect 378 268 381 288
rect 26 194 29 288
rect 682 280 686 284
rect 522 280 526 284
rect 514 280 518 284
rect 506 280 510 284
rect 370 280 374 284
rect 282 280 286 284
rect 274 280 278 284
rect 250 280 254 284
rect 178 280 182 284
rect 42 280 46 284
rect 314 280 325 283
rect 682 191 685 280
rect 522 152 525 280
```

```
rect 506 195 509 280
rect 370 160 373 280
rect 314 152 317 280
rect 274 191 277 280
rect 250 195 253 280
rect 42 160 45 280
rect 834 272 838 276
rect 722 272 726 276
rect 586 272 590 276
rect 570 272 574 276
rect 562 272 566 276
rect 282 272 286 276
rect 834 191 837 272
rect 722 160 725 272
rect 570 136 573 272
rect 282 194 285 272
rect 698 264 702 268
rect 530 264 534 268
rect 514 264 518 268
rect 498 264 502 268
rect 378 264 382 268
rect 362 264 366 268
rect 242 264 246 268
rect 146 264 150 268
rect 82 264 86 268
rect 482 264 493 267
rect 698 195 701 264
rect 530 191 533 264
rect 514 144 517 264
rect 482 136 485 264
rect 362 144 365 264
rect 242 152 245 264
rect 82 194 85 264
rect 322 256 326 260
rect 306 256 310 260
rect 290 256 294 260
rect 234 256 238 260
rect 138 256 142 260
rect 122 256 126 260
rect 114 256 118 260
rect 18 256 22 260
rect 322 191 325 256
rect 290 75 293 256
rect 234 191 237 256
rect 122 128 125 256
rect 114 160 117 256
rect 18 252 21 256
rect 154 248 158 252
rect 74 248 78 252
rect 18 248 22 252
rect 74 191 77 248
rect 18 44 21 248
rect 842 240 846 244
rect 730 240 734 244
rect 690 240 694 244
rect 666 240 670 244
rect 562 240 566 244
rect 490 240 494 244
rect 306 240 310 244
rect 186 240 190 244
rect 162 240 166 244
rect 98 240 102 244
```

```
rect 10 240 14 244
rect 842 127 845 240
rect 730 190 733 240
rect 690 160 693 240
rect 562 188 565 240
rect 490 144 493 240
rect 306 199 309 240
rect 98 160 101 240
rect 10 195 13 240
rect 546 152 549 199
rect 258 160 261 199
rect 50 143 53 199
rect 850 152 853 195
rect 498 128 501 195
rect 346 144 349 195
rect 330 152 333 194
rect 34 144 37 191
rect 578 152 581 190
rect 378 136 381 190
rect 130 160 133 190
rect 714 160 717 188
rect 722 156 726 160
rect 714 156 718 160
rect 690 156 694 160
rect 666 156 670 160
rect 410 156 414 160
rect 370 156 374 160
rect 298 156 302 160
rect 258 156 262 160
rect 130 156 134 160
rect 114 156 118 160
rect 98 156 102 160
rect 42 156 46 160
rect 666 72 669 156
rect 410 79 413 156
rect 850 148 854 152
rect 682 148 686 152
rect 650 148 654 152
rect 610 148 614 152
rect 578 148 582 152
rect 546 148 550 152
rect 522 148 526 152
rect 330 148 334 152
rect 314 148 318 152
rect 298 148 302 152
rect 242 148 246 152
rect 682 74 685 148
rect 650 83 653 148
rect 610 4 613 148
rect 298 12 301 148
rect 514 140 518 144
rect 490 140 494 144
rect 362 140 366 144
rect 346 140 350 144
rect 34 140 38 144
rect 42 140 53 143
rect 42 127 45 140
rect 786 132 790 136
rect 570 132 574 136
rect 482 132 486 136
rect 466 132 470 136
rect 402 132 406 136
```

```
rect 378 132 382 136
rect 26 132 30 136
rect 786 75 789 132
rect 466 72 469 132
rect 402 83 405 132
rect 26 20 29 132
rect 834 124 838 128
rect 826 124 830 128
rect 802 124 806 128
rect 634 124 638 128
rect 586 124 590 128
rect 498 124 502 128
rect 474 124 478 128
rect 386 124 390 128
rect 354 124 358 128
rect 314 124 318 128
rect 186 124 190 128
rect 178 124 182 128
rect 154 124 158 128
rect 138 124 142 128
rect 122 124 126 128
rect 842 124 853 127
rect 34 124 45 127
rect 850 83 853 124
rect 834 4 837 124
rect 826 78 829 124
rect 802 28 805 124
rect 634 75 637 124
rect 586 75 589 124
rect 474 12 477 124
rect 386 75 389 124
rect 354 43 357 124
rect 314 83 317 124
rect 186 74 189 124
rect 178 44 181 124
rect 154 79 157 124
rect 138 75 141 124
rect 34 74 37 124
rect 450 12 453 83
rect 810 4 813 79
rect 602 20 605 79
rect 362 43 365 79
rect 306 36 309 79
rect 626 28 629 78
rect 426 20 429 78
rect 378 28 381 78
rect 338 12 341 78
rect 434 20 437 75
rect 330 36 333 75
rect 482 4 485 74
rect 170 44 173 72
rect 178 40 182 44
rect 170 40 174 44
rect 18 40 22 44
rect 354 40 365 43
rect 330 32 334 36
rect 306 32 310 36
rect 802 24 806 28
rect 626 24 630 28
rect 378 24 382 28
rect 602 16 606 20
rect 434 16 438 20
```

```
rect 426 16 430 20
rect 26 16 30 20
rect 474 8 478 12
rect 450 8 454 12
rect 338 8 342 12
rect 298 8 302 12
rect 834 0 838 4
rect 810 0 814 4
rect 610 0 614 4
rect 482 0 486 4
<< ppcontact >>
rect 458 455 462 479
<< ppdiff >>
rect 458 459 462 482
rect 456 455 464 459
<< nndiff >>
rect 456 517 464 521
rect 458 488 462 517
<< m2contact >>
rect 850 1080 854 1084
rect 722 1080 726 1084
rect 570 1080 574 1084
rect 562 1080 566 1084
rect 554 1072 558 1076
rect 434 1072 438 1076
rect 402 1072 406 1076
rect 226 1072 230 1076
rect 394 1064 398 1068
rect 362 1064 366 1068
rect 210 1064 214 1068
rect 354 1056 358 1060
rect 338 1056 342 1060
rect 138 1056 142 1060
rect 178 1048 182 1052
rect 146 1048 150 1052
rect 706 964 710 968
rect 586 964 590 968
rect 570 964 574 968
rect 418 964 422 968
rect 218 964 222 968
rect 18 964 22 968
rect 626 956 630 960
rect 610 956 614 960
rect 554 956 558 960
rect 386 956 390 960
rect 378 956 382 960
rect 242 956 246 960
rect 850 948 854 952
rect 730 948 734 952
rect 698 948 702 952
rect 602 948 606 952
rect 218 948 222 952
rect 202 948 206 952
rect 194 948 198 952
rect 186 948 190 952
rect 850 940 854 944
rect 682 940 686 944
rect 650 940 654 944
rect 634 940 638 944
rect 170 940 174 944
rect 162 940 166 944
rect 34 940 38 944
```

```
rect 850 932 854 936
rect 842 932 846 936
rect 834 932 838 936
rect 690 932 694 936
rect 538 932 542 936
rect 338 932 342 936
rect 282 932 286 936
rect 250 932 254 936
rect 234 932 238 936
rect 178 932 182 936
rect 154 932 158 936
rect 146 932 150 936
rect 274 924 278 928
rect 258 924 262 928
rect 210 924 214 928
rect 202 924 206 928
rect 858 916 862 920
rect 842 916 846 920
rect 738 916 742 920
rect 722 916 726 920
rect 690 916 694 920
rect 586 916 590 920
rect 562 916 566 920
rect 394 916 398 920
rect 354 916 358 920
rect 578 908 582 912
rect 562 908 566 912
rect 434 908 438 912
rect 402 908 406 912
rect 234 908 238 912
rect 186 908 190 912
rect 178 908 182 912
rect 146 908 150 912
rect 26 908 30 912
rect 842 824 846 828
rect 786 824 790 828
rect 778 824 782 828
rect 690 824 694 828
rect 674 824 678 828
rect 578 824 582 828
rect 554 824 558 828
rect 538 824 542 828
rect 466 824 470 828
rect 418 824 422 828
rect 266 824 270 828
rect 234 824 238 828
rect 194 824 198 828
rect 170 824 174 828
rect 18 824 22 828
rect 754 816 758 820
rect 746 816 750 820
rect 722 816 726 820
rect 626 816 630 820
rect 586 816 590 820
rect 386 816 390 820
rect 274 816 278 820
rect 266 816 270 820
rect 226 816 230 820
rect 218 816 222 820
rect 34 816 38 820
rect 602 808 606 812
rect 594 808 598 812
```

```
rect 482 808 486 812
rect 434 808 438 812
rect 330 808 334 812
rect 298 808 302 812
rect 226 808 230 812
rect 210 808 214 812
rect 82 808 86 812
rect 74 808 78 812
rect 66 808 70 812
rect 50 808 54 812
rect 450 800 454 804
rect 282 800 286 804
rect 258 800 262 804
rect 162 800 166 804
rect 154 800 158 804
rect 146 800 150 804
rect 34 792 38 796
rect 26 792 30 796
rect 802 708 806 712
rect 746 708 750 712
rect 738 708 742 712
rect 730 708 734 712
rect 586 708 590 712
rect 570 708 574 712
rect 338 708 342 712
rect 322 708 326 712
rect 794 700 798 704
rect 634 700 638 704
rect 602 700 606 704
rect 530 700 534 704
rect 378 700 382 704
rect 298 700 302 704
rect 266 700 270 704
rect 250 700 254 704
rect 226 700 230 704
rect 202 700 206 704
rect 10 700 14 704
rect 282 692 286 696
rect 754 684 758 688
rect 594 684 598 688
rect 546 684 550 688
rect 362 684 366 688
rect 314 684 318 688
rect 306 684 310 688
rect 266 684 270 688
rect 10 684 14 688
rect 778 676 782 680
rect 746 676 750 680
rect 578 676 582 680
rect 554 676 558 680
rect 506 676 510 680
rect 498 676 502 680
rect 482 676 486 680
rect 186 676 190 680
rect 82 676 86 680
rect 50 676 54 680
rect 42 676 46 680
rect 202 668 206 672
rect 34 668 38 672
rect 746 584 750 588
rect 626 584 630 588
rect 594 584 598 588
```

```
rect 506 584 510 588
rect 306 584 310 588
rect 170 584 174 588
rect 26 584 30 588
rect 634 576 638 580
rect 618 576 622 580
rect 226 568 230 572
rect 210 568 214 572
rect 178 568 182 572
rect 74 568 78 572
rect 42 568 46 572
rect 10 568 14 572
rect 330 560 334 564
rect 322 560 326 564
rect 250 560 254 564
rect 26 560 30 564
rect 770 552 774 556
rect 754 552 758 556
rect 554 552 558 556
rect 482 552 486 556
rect 338 552 342 556
rect 306 552 310 556
rect 298 552 302 556
rect 146 552 150 556
rect 130 552 134 556
rect 42 552 46 556
rect 730 544 734 548
rect 610 544 614 548
rect 586 544 590 548
rect 482 544 486 548
rect 450 544 454 548
rect 154 544 158 548
rect 618 536 622 540
rect 594 536 598 540
rect 522 536 526 540
rect 466 536 470 540
rect 754 528 758 532
rect 746 528 750 532
rect 626 528 630 532
rect 362 528 366 532
rect 346 528 350 532
rect 330 528 334 532
rect 322 528 326 532
rect 274 528 278 532
rect 746 444 750 448
rect 642 444 646 448
rect 626 444 630 448
rect 474 444 478 448
rect 466 444 470 448
rect 314 444 318 448
rect 178 444 182 448
rect 162 444 166 448
rect 58 444 62 448
rect 10 444 14 448
rect 786 436 790 440
rect 602 436 606 440
rect 554 436 558 440
rect 490 436 494 440
rect 314 436 318 440
rect 146 436 150 440
rect 138 436 142 440
rect 114 436 118 440
```

```
rect 106 436 110 440
rect 90 436 94 440
rect 26 436 30 440
rect 18 436 22 440
rect 610 428 614 432
rect 538 428 542 432
rect 522 428 526 432
rect 386 428 390 432
rect 354 428 358 432
rect 330 428 334 432
rect 154 428 158 432
rect 82 428 86 432
rect 74 428 78 432
rect 762 420 766 424
rect 586 420 590 424
rect 578 420 582 424
rect 306 420 310 424
rect 178 420 182 424
rect 162 420 166 424
rect 50 420 54 424
rect 34 420 38 424
rect 754 412 758 416
rect 570 412 574 416
rect 546 412 550 416
rect 298 404 302 408
rect 282 404 286 408
rect 130 404 134 408
rect 682 396 686 400
rect 650 396 654 400
rect 618 396 622 400
rect 562 396 566 400
rect 514 396 518 400
rect 490 396 494 400
rect 482 396 486 400
rect 434 396 438 400
rect 290 396 294 400
rect 802 388 806 392
rect 634 388 638 392
rect 610 388 614 392
rect 162 388 166 392
rect 146 388 150 392
rect 26 388 30 392
rect 522 304 526 308
rect 498 304 502 308
rect 338 304 342 308
rect 314 304 318 308
rect 298 304 302 308
rect 282 304 286 308
rect 34 304 38 308
rect 26 304 30 308
rect 506 296 510 300
rect 298 296 302 300
rect 58 296 62 300
rect 546 288 550 292
rect 530 288 534 292
rect 26 288 30 292
rect 682 280 686 284
rect 522 280 526 284
rect 514 280 518 284
rect 506 280 510 284
rect 370 280 374 284
rect 282 280 286 284
```

```
rect 274 280 278 284
rect 250 280 254 284
rect 178 280 182 284
rect 42 280 46 284
rect 834 272 838 276
rect 722 272 726 276
rect 586 272 590 276
rect 570 272 574 276
rect 562 272 566 276
rect 282 272 286 276
rect 698 264 702 268
rect 530 264 534 268
rect 514 264 518 268
rect 498 264 502 268
rect 378 264 382 268
rect 362 264 366 268
rect 242 264 246 268
rect 146 264 150 268
rect 82 264 86 268
rect 322 256 326 260
rect 306 256 310 260
rect 290 256 294 260
rect 234 256 238 260
rect 138 256 142 260
rect 122 256 126 260
rect 114 256 118 260
rect 18 256 22 260
rect 154 248 158 252
rect 74 248 78 252
rect 18 248 22 252
rect 842 240 846 244
rect 730 240 734 244
rect 690 240 694 244
rect 666 240 670 244
rect 562 240 566 244
rect 490 240 494 244
rect 306 240 310 244
rect 186 240 190 244
rect 162 240 166 244
rect 98 240 102 244
rect 10 240 14 244
rect 722 156 726 160
rect 714 156 718 160
rect 690 156 694 160
rect 666 156 670 160
rect 410 156 414 160
rect 370 156 374 160
rect 298 156 302 160
rect 258 156 262 160
rect 130 156 134 160
rect 114 156 118 160
rect 98 156 102 160
rect 42 156 46 160
rect 850 148 854 152
rect 682 148 686 152
rect 650 148 654 152
rect 610 148 614 152
rect 578 148 582 152
rect 546 148 550 152
rect 522 148 526 152
rect 330 148 334 152
rect 314 148 318 152
```

```
rect 298 148 302 152
rect 242 148 246 152
rect 514 140 518 144
rect 490 140 494 144
rect 362 140 366 144
rect 346 140 350 144
rect 34 140 38 144
rect 786 132 790 136
rect 570 132 574 136
rect 482 132 486 136
rect 466 132 470 136
rect 402 132 406 136
rect 378 132 382 136
rect 26 132 30 136
rect 834 124 838 128
rect 826 124 830 128
rect 802 124 806 128
rect 634 124 638 128
rect 586 124 590 128
rect 498 124 502 128
rect 474 124 478 128
rect 386 124 390 128
rect 354 124 358 128
rect 314 124 318 128
rect 186 124 190 128
rect 178 124 182 128
rect 154 124 158 128
rect 138 124 142 128
rect 122 124 126 128
rect 178 40 182 44
rect 170 40 174 44
rect 18 40 22 44
rect 330 32 334 36
rect 306 32 310 36
rect 802 24 806 28
rect 626 24 630 28
rect 378 24 382 28
rect 602 16 606 20
rect 434 16 438 20
rect 426 16 430 20
rect 26 16 30 20
rect 474 8 478 12
rect 450 8 454 12
rect 338 8 342 12
rect 298 8 302 12
rect 834 0 838 4
rect 810 0 814 4
rect 610 0 614 4
rect 482 0 486 4
<< pwell >>
rect 453 450 467 485
<< nncontact >>
rect 458 489 462 521
<< nwell >>
rect 453 485 467 526
<< labels >>
rlabel metal1 -24 908 -21 911 0 select
rlabel metal1 -24 676 -21 679 0 load
rlabel metal1 -24 1072 -21 1075 0 in[0]
rlabel metal1 -24 412 -21 415 0 in[1]
rlabel metal1 -24 280 -21 283 0 in[2]
rlabel metal1 -24 264 -21 267 0 in[3]
```

```
rlabel metal1 -24 272 -21 275 0 in[4]
rlabel metal1 -24 948 -21 951 0 in[5]
rlabel metal1 -24 1080 -21 1083 0 in[6]
rlabel metal1 -24 1048 -21 1051 0 in[7]
rlabel metal1 -24 708 -21 711 0 in[8]
rlabel metal1 -24 24 -21 27 0 in[9]
rlabel metal1 -24 536 -21 539 0 in[10]
rlabel metal1 -24 388 -21 391 0 in[11]
rlabel metal1 -24 544 -21 547 0 in[12]
rlabel metal1 -24 8 -21 11 0 in[13]
rlabel metal1 -24 956 -21 959 0 in[14]
rlabel metal1 -24 576 -21 579 0 in[15]
rlabel metal1 893 800 896 803 0 outA[0]
rlabel metal1 893 264 896 267 0 outA[1]
rlabel metal1 893 16 896 19 0 outA[2]
rlabel metal1 893 956 896 959 0 outA[3]
rlabel metal1 893 404 896 407 0 outA[4]
rlabel metal1 893 1048 896 1051 0 outA[5]
rlabel metal1 893 908 896 911 0 outA[6]
rlabel metal1 893 1056 896 1059 0 outA[7]
rlabel metal1 893 156 896 159 0 outA[8]
rlabel metal1 893 24 896 27 0 outA[9]
rlabel metal1 893 0 896 3 0 outA[10]
rlabel metal1 893 428 896 431 0 outA[11]
rlabel metal1 893 296 896 299 0 outA[12]
rlabel metal1 893 692 896 695 0 outA[13]
rlabel metal1 893 568 896 571 0 outA[14]
rlabel metal1 893 1080 896 1083 0 outA[15]
rlabel metal1 -24 40 -21 43 0 clk_b
rlabel metal1 -24 124 -21 127 0 reset_b
rlabel metal1 893 1064 896 1067 0 outB[0]
rlabel metal1 893 256 896 259 0 outB[1]
rlabel metal1 893 668 896 671 0 outB[2]
rlabel metal1 893 248 896 251 0 outB[3]
rlabel metal1 893 816 896 819 0 outB[4]
rlabel metal1 893 924 896 927 0 outB[5]
rlabel metal1 893 916 896 919 0 outB[6]
rlabel metal1 893 792 896 795 0 outB[7]
rlabel metal1 893 560 896 563 0 outB[8]
rlabel metal1 893 8 896 11 0 outB[9]
rlabel metal1 893 808 896 811 0 outB[10]
rlabel metal1 893 40 896 43 0 outB[11]
rlabel metal1 893 584 896 587 0 outB[12]
rlabel metal1 893 32 896 35 0 outB[13]
rlabel metal1 893 1072 896 1075 0 outB[14]
rlabel metal1 893 388 896 391 0 outB[15]
rlabel metal1 -24 109 -16 117 0 Vdd
rlabel metal1 -24 51 -16 59 0 GND
rlabel metal1 888 51 896 59 0 GND_2_
rlabel metal1 888 109 896 117 0 Vdd_2_
rlabel metal1 -24 167 -16 175 0 GND_3_
rlabel metal1 888 167 896 175 0 GND_4_
rlabel metal1 -24 225 -16 233 0 Vdd_3_
rlabel metal1 888 225 896 233 0 Vdd_4_
rlabel metal1 -24 315 -16 323 0 GND_5_
rlabel metal1 888 315 896 323 0 GND_6_
rlabel metal1 -24 373 -16 381 0 Vdd_5_
rlabel metal1 888 373 896 381 0 Vdd_6_
rlabel metal1 -24 455 -16 463 0 GND_7_
rlabel metal1 888 455 896 463 0 GND_8_
rlabel metal1 -24 513 -16 521 0 Vdd_7_
rlabel metal1 888 513 896 521 0 Vdd_8_
```

```
rlabel metal1 -24 595 -16 603 0 GND_9_
rlabel metal1 888 595 896 603 0 GND_10_
rlabel metal1 -24 653 -16 661 0 Vdd_9_
rlabel metal1 888 653 896 661 0 Vdd_10_
rlabel metal1 -24 719 -16 727 0 GND_11_
rlabel metal1 888 719 896 727 0 GND_12_
rlabel metal1 -24 777 -16 785 0 Vdd_11_
rlabel metal1 888 777 896 785 0 Vdd_12_
rlabel metal1 -24 835 -16 843 0 GND_13_
rlabel metal1 888 835 896 843 0 GND_14_
rlabel metal1 -24 893 -16 901 0 Vdd_13_
rlabel metal1 888 893 896 901 0 Vdd_14_
rlabel metal1 -24 975 -16 983 0 GND_15_
rlabel metal1 888 975 896 983 0 GND_16_
rlabel metal1 -24 1033 -16 1041 0 Vdd_15_
rlabel metal1 888 1033 896 1041 0 Vdd_16_
<< end >>
```

VITA

Subramanian Sivaramakrishnan

Candidate for the Degree of

Master of Science

Thesis: AN IMPLEMENTATION OF A SPATIAL DATABASE SYSTEM

Major Field: Computer Science

Biographical:

Personal Data: Born in Dubai, United Arab Emirates, on July 4, 1969, son of N. Sivaramakrishnan and Visalakshy Sivaramakrishnan.

Education: Received Bachelor of Engineering Degree in Computer Engineering from Madurai Kamaraj University at Madurai, India in June 1990; completed requirements for the Master of Science degree at Oklahoma State University in December 1993.

Experience: Teaching Assistant, Computer Science Department, Oklahoma State University, August 1991 to December 1993.